THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

A METHODOLOGY FOR THE DISTRIBUTION OF DATABASES

Mourad Oulid-Aissa

CRL-TR-26-84

Under the Direction of
Professor Keki B. Irani

APRIL 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]This research was supported by the Department of the Army, Ballistic Missile Defense Advanced Technology Center, Rome Air Development Center, and the Defense Mapping Agency under Contract F30602-80-C-0173. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the funding agency.

ABSTRACT

A METHODOLOGY FOR THE DISTRIBUTION OF DATABASES

by

Mourad Oulid-Aissa

Chairman: Keki B. Irani

This research proposes an approach to distribute data that feature semantic cross-references in a computer network. The unit of distribution is the functional dependency (FD).

Firstly, each scheduled query is assigned a set of cross-referencing data units (CRDUs) that will normally be accessed, at possibly different sites, to process it. Each CRDU is associated with one FD. The selected FDs are mapped to a so-called query envelope. The selection of a query envelope is based on semantic integrity and on the minimization of the total "volume" of the instances of the FDs belonging to the envelope. This problem, identified as the envelope optimization problem, is presented and analyzed in a formal framework, and is solved with efficient graph algorithms.

Secondly, the CRDUs corresponding to the FDs of all the envelopes are distributed so as to minimize the operational communication cost. The issue of preservation of the database consistency is mentioned. Further, the following modeling points are considered: (i) The cross-references of physically distant CRDUs, which produce file-to-file inter-site communication in addition to the usual file-to-user communication, (ii) the partitioning of the set of

CRDUs, (iii) the materialization of the CRDUs associated with one envelope, in case of duplication, (iv) the flow capacity constraints.

The problem of the distribution of CRDUs is modeled in the form of a manageable multi-commodity mixed-integer program, with quadratic cost and linear constraints. Computational techniques and numerical experience are discussed. Through experimental investigations, the research draws some practical conclusions about the distribution of CRDUs in a computer network. The effect of the reducing and assembly rates over the simultaneous referencing of CRDUs, and over data redundancy, is investigated. Further, the effect of the updating rates over cross-referencing, and over data redundancy, is investigated. The results suggest that a look-up of the user-provided data can indicate whether standard data distribution techniques, instead of CRDU distribution techniques, can be used.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

The goal of this thesis is to outline a detailed procedure which can be applied to the design of distributed databases. In this chapter, the main problem of interest is introduced. Research objectives are discussed in Section 1.5. Initially described are the notions of "distributed system" (Section 1.1) and "user environment" (Section 1.2). The principles of distributed query processing are discussed in Section 1.3. Section 1.4 introduces the notion of a "distributed database" and states the central problem of interest. Section 1.5 provides some key arguments to support our contention that the design methodologies of database distribution available at the time of this writing suffer from shortcomings. Section 1.6 emphasizes one important issue in the research, the definition of a distribution unit. Finally, the research objectives are stated in Section 1.7 in informal terms.

1.1. The physical environment

Our work applies to a physical environment referred to as a *distributed system**. A distributed system is characterized by the placement of processing and control functions** at different computing *sites,* as opposed to a *centralized system* in which all the processing and control functions are located in a main computing center. A single, usually powerful, computing unit is enough to support a centralized system, however, a distributed system usually implies the existence of an underlying *computer network.* A computer network can be visual-

*Technical words in italic, in Chapters 1 and 2, are defined in Appendix A.

**A "function" can be visualized as a software module. We refer to functions that are part of a distributed management system, and in particular, the functions applied to the control and processing of transactions.

ized as a set of interconnected sites or *nodes*. Thus, a computer network consists of a set of computing units that can exchange information between each other through a common communication medium (usually a network of tightly connected processors, i.e. connected by telephone channels, or loosely connected processors, i.e. communicating through a broadcasting medium [BOOT 81, DORA 77]). A distributed system can also be supported by a *database machine* or some more general parallel computing machines. A fundamental property of a distributed system is its ability to remain operative for most users (or most jobs), in the event of occurrence of localized failures of some nodes, or of communication disruptions. In this context it is assumed that a computer network stands for a set of *autonomous* computers together with their interconnecting links. (However, the material of this thesis is also relevant for the design of distributed systems conceived on one or several database machines). Computers which are said to be autonomous are logically equal [TANE 81], i.e. they participate within a distributed system in an "equal partnership" relationship. This type of physical environment excludes any hierarchical or master-slave relationship between any two computers. Furthermore, the computers of such an environment, sometimes called a *horizontal* distributed system, need not have the same physical characteristics. In addition to considering a distributed system, we will focus on the usage of such a system in the context of database applications.

1.2. The user environment

The user environment is characterized by the nature of the information processing applications. In this context, the user environment translates into a database that is to be processed through intensive cross-referencing of data. Distributed databases are relevant for the support of such organizations as government agencies, communication industries, transportation companies, and the military, to name just a few. In general, the type of database which we wish to consider is one that contains a huge amount of intricately related "text" or numerical data. The information that it represents is under periodic querying and updating, and in general, each query may require more than a simple reading operation. A query may indeed involve complex *cross-referencing* of data. Users originating such queries from each site of the computer network form a homogeneous usage environment in the sense that they all belong to the same organization and that the entire database appears to them as if it were centralized. It is assumed however that there may exist some typical patterns for the queries and updates initiated from any specific site. The typical transactions are referred to as the scheduled queries and updates*. Hence, although the information available through the database is accessible by any user from any site, one site may be associated with users with specialized interests who originate certain types of transactions more frequently than other types of transactions. The data needs of a user constitute the *view* of that user. The user environment to be considered consists of one homogeneous distributed database, and a set of users' views, where a user is identified with one computer or site of the underlying computer network of the distributed system. We now wish to refine the description of the user environment and to introduce some technical notions of distributed query processing.

1.3. Distributed query processing

The concept of distribution implies more than a mere geographical distancing of data. Indeed, it presupposes some strong features regarding the strategies of query processing. Normally, a query is processed in three steps. During the first step, any query issued at a user-site, by virtue of its qualification clauses, leads to some extensive local processing on a set of initially accessed relations**. The relevant relations are usually reduced in size by a sequence of *restriction* and *projection* operations. The second step involves the complex *cross-referencing* of the reduced relations. Those relations which are semantically dependent, are involved in a mutual effort to further reduce one another in order to eliminate as much

*As opposed to ad-hoc queries and updates.

**Since it is common to use the terminology proper to relational database, the word *relation* will often be favored over the less descriptive word *file*.

irrelevant data, for the query, as possible† [CHUN 83]. In practice, this is usually accomplished by performing a sequence of *semi-joins* between distant relations††. Data cross-referencing in distributed query processing is associated with the so-called *inter-site* or *file-to-file* communication. This type of communication usually incurs a relatively large flow of data volume, also known, for obvious reasons, as the *reducing* flow in the computer network‡. When the reduction step is completed, the copies of the reduced relations contain the information requested by the query, and are ready to be shipped to and *assembled* at the user-site to produce a final answer‡‡. This third and last step of distributed query processing is referred to as the *file-to-user* communication, and is associated with the so-called *assembly* flow in the computer network*. At the design stage, the data units which are subject to distribution are referred to as *cross-referencing data units* (CRDUs), because in the operational stage of the distributed database, they will assume the form of relational tables or fragments thereof**, and will be submitted to cross-referencing during the distributed query processing sessions. The objective of this research is to find an efficient and formal approach to the problem of defining and distributing CRDUs.

1.4. The problem of database distribution

Although the problem of "where to place the data" can sometimes be solved pragmatically, it is nevertheless a non-trivial one mainly because of data dependence. Further, data distribution may rely in many ways on formal optimization techniques.

It is usually not practical to replicate and store a whole database at each site of a network, particularly in the case of a very large database. Data units which are semantically

†E.g. if they have one or more common attributes.

††Located at different sites.

‡The words *reducing* and *file-to-file* will be associated with a logical operation such as a semi-join, with the cross-referencing communication process established between two relations, or with the resulting flow in the network.

‡‡The assembling operation may be performed at a site distinct from the user-site. However, there seems to be no advantage in doing so.

*The remark made in footnote ‡ applies also for the terms *assembly* and *file-to-user*.

**E.g. after the application of projection and restriction on some larger relations yet to be defined.

dependent on one another may have to be stored at different sites, and may thus require some information to be represented with partial *redundancy.* A single transaction originated at one site may therefore cause some interlocation accesses to take place between several sites other than the user-site. Such a database is called a *distributed database.* Hence, a distributed database is characterized by the storage of data units, organized into files, at several sites of a computer network. Any data unit may be stored in at least one site and may be accessed by any other site when needed.

In this context, the overall operating cost of the distributed database results primarily from the communication cost per unit time. Given a network of computers that share common data units, the problem of finding a placement of the data units over the computing sites, such that this placement yields a minimum overall operating cost, while satisfying a set of physical and logical constraints, has been identified in the literature as the "data allocation" problem. Usually, whether multiple copies of data units are allowed or not is stated explicitly.

This research addresses a larger problem; that of defining, selecting and distributing CRDUs so that relations can be synthesized at each site. This problem is identified as the *database distribution problem.* Thus, data allocation can be considered to be a subproblem of the database distribution problem. Note that the word "distribution" is used, rather than the word "allocation". In our terminology, "distribution" includes the possibility of keeping duplicate copies of any one of the CRDUs. In the section that follows two issues which motivate further research in the solution of the database distribution problem are indicated.

1.5. Research motivations

For some organizations, a specific distribution of data may naturally be dictated by the rigid nature of the applications associated with the sites of a network [BOOT 81]. However, if the database to be designed is not initially partitioned, and if no a priori distribution naturally emerges, then the database distribution problem needs to be addressed, to some extent, in optimization terms. Further, the most critical issue to consider when distributing data in a computer network is the definition of a unit of distribution. This latter issue is discussed at more length in Chapter 2. The choice of a unit of distribution will eventually affect the performance of a distributed database.

The traditional methodology [BUCK 79] applied to the database distribution problem, consists of dealing separately with the two problems of defining the database files and of allocating them to the computers of the network. Hence, the allocation phase usually rests upon the assumption that the data are already organized into files. The methodology applied to define those files makes use of centralized database design techniques. That design approach is inefficient because measures of performance, mainly the traffic cost associated with distributed query processing mechanisms, are not taken into consideration at the time when the initial files are defined. In general, the distributed database design methodologies available at the time of this writing are based on the pragmatic notion that file definition and database distribution are two independent problems [BUCK 79, ESWA 74]. It must be realized that this approach is a simplified solution to the general database distribution problem and needs to be reviewed and extended. The underlying methodology of this work emphasizes, to some extent, a synthesis approach of design. The proposed design process starts with the definition of the CRDUs, which may be looked at as the elementary building blocks of some, yet undefined, relations, and it ends with the definition and usage of those relations in the distributed system environment.

The latter paragraph contains some remarks about the methodology of distributing databases. The next observation refers specifically to current data allocation models and their solution techniques. In these models, it appears that the assumption of data independence is routinely made. This assumption implies that the distributed query processing mechanisms are exclusively modeled as sequences of *file-to-user* accesses [RAMA-b-79]. Indeed, this assumption of data independence implies that any query can be solved by sending a complete answer from a distant site (that possesses the required information) to the site where the query is issued. Therefore, the simplest models of data allocation follow standard mathematical structures, similar to those of the "facility-location" type problems in operations research. However, the data independence assumption (which is a prerequisite for those models) cannot realistically be applied to complex databases that make use of relational database techniques since it is inadequate to represent distributed query processing mechanisms, i.e. the cross-referencing of data units located at different sites, as was noted previously. The issue of cross-referencing in distributed query processing will be discussed in more detail in the next chapter. Furthermore, the data independence assumption prevents us from investigating the effect on the traffic of the update transactions, which are originated by the system (as opposed to those originated by the user), to preserve database consistency. Updates and update side-effects will be treated in Section 4.5. Thus, if optimization is to be included in a methodology for database distribution, some basic semantic issues must be considered, and therefore, modifications must be made on current mathematical programming models of data allocation.

1.6. Cross-referencing data units (CRDUs)

In this section, the term "cross-referencing data unit" is given a precise definition. The soundness of such a definition is argued.

A transaction originated by a user at some site of the network is really an access of data-items or *attributes*. Therefore, instead of defining a CRDU to be a file or a relation, a natural approach might be to associate a CRDU with an attribute. However, attributes, as such, are meaningless from a semantic point of view. The notion of dependency among attributes, which is central to guarantee the correctness of transaction processing, suggests that groups of attributes can be used to define CRDUs. To ensure query correctness, several rules have been developed in the literature. These rules hold for semantic constraints of the functional type* known as *functional dependencies* (FD) [ARMS 74]. It is convenient to identify a CRDU, i.e. an atomic data unit which can be subject to distribution, with an FD. Semantic constraints of the functional type, or FDs, are chosen for the following reasons. (1) FDs are sufficient to determine whether a data access leads to a semantically correct answer, for a query, or not. (2) They are simple and easily identifiable by the user. (3) They are the building elements of the synthetic approach to construct *logical database schemata* for the three major models, the "network", the "hierarchical", and the "relational models". It has been shown that FDs can be used to design logical databases for any of the three models mentioned [BERN 76, DELO 78, HOUS 79]. Further support for the choice of FDs to stand for CRDUs will be provided in Section 2.2.

The *relational model* is selected because it is the best adapted to current distributed query processing terminologies as discussed in Chapter 2. However, this does not presuppose any restriction as far as the local physical database implementations are concerned. In this context, the CRDUs are the physical counterparts of FDs, and can be visualized as particular types of *relational tables* featuring at least two columns of attribute values. In the remainder of this thesis, FD is used to denote a functional dependency as an abstract concept. A CRDU refers to the physical instance of an FD.

Algorithms for synthesizing relational tables [WANG 75, BERN 76, KAMB 78] permit the construction of relations at each site after the CRDUs mapped into FDs are distributed. Thus, under our definition of the CRDUs, it will be possible to define a methodology which introduces optimization early in the solution of the database distribution problem.

*This choice is made without any loss of generality because dependencies between attributes that are not of the functional type can be modeled with artificial FDs called ***non-functional relationships***. If the concept of non-functional relationships [BERN 76] is considered to be too controversial for some readers, another argument supporting the use of FDs is that it is widely acknowledged that regular FDs augmented with the related notion of *join-dependency* (defined in Section 3.1) are enough to capture the semantics of most practical databases [BEER 81].

1.7. Objectives

The problem addressed in this thesis is now broadly stated. A computer network which consists of computer sites and of communication links is available. The computer network has a predefined topology and the links have fixed channel capacities. It is equipped with a horizontal distributed system in the sense of Section 1.1. A distributed database is to be designed for the benefit of an organization of users in the sense of Section 1.2, and a universe of data-items is provided by the organization together with a set of FDs. This organization also provides specifications for every transaction, query and update, at each site, together with the occurence frequencies of those transactions and some statistical information about "raw" data-volumes* with respect to FDs and attributes.

The problem objectives are the following. (1) Model and select a set of CRDUs which may be subject to distribution without any loss of the over-all information, and which will allow simple and efficient file-to-file accesses in the operating phase of the distributed database. Determine how the CRDUs are affected by updates. (2) Represent the problem of organizing CRDUs in the computer network in the form of a manageable mathematical optimization model. The organization of the CRDUs must capture** (i) the cross-referencing characteristics of distributed query processing, (ii) the *clustering* of the CRDUs, e.g. whether they belong to a common relational table or not, and the resulting effect on the assembly and reducing flows of data-volume, and (iv) the *materialization,* in the static sense, of the CRDUs i.e. the choice of relevant instances of CRDUs to seek access to, for each query, when there are duplicates. (3) Explore and implement efficient numerical techniques to solve the proposed model of (2) so as to minimize the operational communication cost.

The next chapter surveys related studies in the literature, and develops some background for the research.

*See Section 4.4.

**The following issues will be explained in more details in the next chapter.

CHAPTER 2

BACKGROUND

In this chapter the conceptual foundations needed to achieve the objectives of Section 1.5 are laid out. Relevant issues are discussed in light of related studies in the literature. At the same time, some basic assumptions are made. The relational model is first introduced (Section 2.1). The usage of functional dependencies in database schema design is discussed (Section 2.2), and means to depict functional dependencies are reviewed (Section 2.3). Distributed query processing is described (Section 2.4), followed with a survey of earlier data allocation models (Section 2.5).

2.1. The relational model

Some basic technical terms of relational database modeling are introduced. The relational model [CODD 72] has been the object of very intensive and rich studies. It is legitimate to claim that database theory, as it is being developed now, was born out of the relational model. What makes it attractive is its nonprocedural orientation. The definition of a relational database will not be expanded on; comprehensive presentations can be found in [CODD 79, CODD 72, DATE 78, ULLM 80] to name just a few sources. A *relation* or *relational table* will be referred to as a subset of the arbitrarily ordered cartesian product of a list of *domains*. A domain is itself a value set for one or several *attribute types* (the "atomic" data-item types). A relation is a set of *tuples*. Thus, a tuple represents a possible value for a specific list of attribute types. That list constitutes the heading of a relation. Well-defined operations can be applied to relations to reduce them or to combine them together. These operations are used to define the *relational algebra* [CODD 79, CODD 72, DATE 78]. More will be said below about the use of these operations in conjunction with distributed query processing policies.

In the next section, *functional dependencies* are discussed. They can be visualized as special types of relations that can be used as building blocks to generate more complete relations. The process of defining relations from FDs is known as the synthetic approach of logical schema design.

2.2. Functional dependency: A tool for logical schema design

The *functional dependency* (FD) semantic constraints serve as building units in database design. FDs were first mentioned in connection with the *normalization* of relational tables as a measure to avoid update anomalies [CODD 72]. Axioms pertaining to FDs were established in [ARMS 74] as a foundation for a formal description of database semantics. After the first definition of *third normal form* was proposed in [CODD 72], there followed the concept of *multivalued dependency* (MVD) [FAGI 77], implying stricter normal form definitions. Database theorists recognized early the interest for *synthesizing* schemata from functional dependencies [WANG 75, BERN 76]. The synthesis approach consists of building normalized relational tables given an initial set of FDs. However, in [KAMB 78], the algorithms presented in [WANG 75, BERN 76] were shown to feature some weaknesses. In this latter work, a new algorithm for synthesizing relational tables was presented. As mentioned in [KAMB 78], the reason for considering FDs exclusively, and not multivalued dependencies, in the context of relational tables synthesis, stems from what we might call the universality property of FDs. To quote Kambayashi [KAMB 78]:

An FD $X \rightarrow Y$ holds in any set of attributes that contains $X \bigcup Y$ under the uniqueness assumption discussed in [BERN 76]; an MVD is defined together with the set of attributes where it holds.

This is why MVDs are considered primarily in the context of a *decomposition* approach of schema design [BEER 80, ZANI 81] rather than a *synthetic* approach. A representation of a database as a set of "3NF"- normalized relational tables is not unique [LOZI 78]. Heuristics aiming at improving the performance of a *centralized* relational database were devised as an extension to the synthetic approach [LOZI 78].

The relational model is not the only logical database model which gains to make use of FDs. FDs have been shown to serve in the conversion from a hierarchical model to a relational model [DELO 78]. Also, they have been used to construct logical database schemata described with the network model [HOUS 79]. In [HOUS 79], operations defined on FDs are used to "navigate" through a schema and to describe transactions in a formal way. However, those "navigation" mechanisms do not always capture a correct query processing since so-called *lossy* accesses, as described in [AHO 77], may sometimes occur. It has been argued that a set of FDs together with the notion of *lossless join* (to be discussed in the next section), is enough to capture the semantics of most practical databases [BEER 81]. Further, the important problem resulting from *deletion cascades* [HOUS 79] is most simply observed through FD-modeling. Deletion side-effects associated with consistency preservation can assume a critical role in database distribution. These particular aspects of query correctness, i.e. data *integrity* and data *consistency* have been avoided in previous data allocation methods [PALM 79, BUCK 79], and in mathematical file allocation models [CASE 72, CHU 73, ESWA 74, MAHM 76, MORG 77], mainly through the assumption of data independence. The preceding arguments support further the claim, made in Chapter 1, that it is indeed appropriate and quite general to retain FDs as the sole low-level semantic modeling tools.

In the next section, issues pertaining to the depiction of a *minimum cover* of FDs, i.e. a smallest set of FDs which is semantically equivalent to a given set of FDs, are discussed.

2.3. Minimum depiction of functional dependencies

The main idea of this section is that the non-redundant depiction of a set of FDs by graphical means is a convenient way to arrive at the representation of a minimum cover of that set.

In [AHO 77, BISK 79, LOZI 80], similar results concerning *lossless joins* of relations have been reported. The main result of interest to us is summarized in Theorem 1 (Section 4.1) which combines two corollaries presented in [BISK 79]. Corollaries 1 and 3, and Proposition 7 extend this result to relational schemes involving FDs uniquely. The notion of lossless join is useful to determine whether or not a query solution respects the database integrity. We are not considering the correctness of actual query processing steps, since we assume that appropriate processing strategies are available, but rather the validity of the so-called query envelopes [CCA 80]. In this work, the envelope of a query is informally defined to be a portion of the data which is sufficient for answering the query. A query envelope consists of a set of FDs that features the lossless join property. Since, for each query, the choice of an envelope is not unique, optimization will be used to determine a "good" query envelope. The envelope optimization problem is not unlike the problems considered in [LOZI 78, LOZI 80] for a centralized database. In [LOZI 78] the author considers possible improvements of Bernstein's synthesis algorithm [BERN 76] to enhance database performance. In [LOZI 80], given a set of relations, the author looks for efficient query procedures. We borrow the notion of F-graph to depict FDs, and the notion of a *link* to extract a relevant portion of a database for a query, from [LOZI 80]. (Although this latter notion will be refined and adapted to our specific needs). Because of the "distributed" nature of our envelope optimization problem, our assumptions and solution techniques are different from those proposed in [LOZI 80].

The methodology described in [LOZI 80] begins with an initial F-graph (Section 3.1.2) to depict FDs, but no mention is made as to how such a graph could be obtained without *redundancy*. The generation of an initial F-graph from a user-provided F-graph is one important issue to address, especially in the context of automatic schema design. A set of reducing rules for F-Graphs based on Propositions 1-3 (Section 4.2) are provided to eliminate redundancy systematically. The approach which consists of building reduced F-graphs is justified by Propositions 5-7 (Section 4.2), in light of previous results on minimum FD covers reported in [MAIE 80]. It is shown that Algorithm *Reduce* of Section 4.1 which constructs a non-redundant F-Graph to depict FDs, leads to the same result and is at most as complex as [MAIE 80]'s algorithm which derives *LR-minimum* covers of FDs.

The discussion on FDs and the means to represent them suggests that they can be utilized to describe a user's *view* of its data [FEDA 81] and also a system view of the data (if the FDs are rearranged to eliminate excessive redundancy). The focus will be placed on a low level of abstraction corresponding roughly to the "instance operations level" of [NAVA 76], and not in consideration of mappings from *external schemata* to *conceptual schema* as in [FEDA 81], although the importance of this latter issue at a higher semantic level is recognized.

Determining an envelope of lossless FDs for a query constitutes a starting point from where actual query processing *sequences* may be derived [CHUN 83]. In the next section distributed query processing strategies are discussed.

2.4. Distributed query processing

It is essential for comprehension of the proposed model (Chapters 4 and 5) to outline the characteristics of distributed query processing [HEVN 79, WONG 77, CCA 80, CHUN 83]. The first issue associated with the solution of a distributed query is the localization, through a data directory, of the relevant data needed [CHU 76]. It is assumed that such directories are available at each site [CHU 76]. Since data may be replicated at different sites, a specific non-redundant instance of data must be determined for each query. This instance of data is known as a *static materialization* [CCA 80, HEVN 79]. A static materialization refers to a physical representation of the query envelope which itself refers to a smallest portion of the conceptual schema relevant to solve the query. A materialization can be obtained by table look-up [CCA 80] or can be determined dynamically during query processing [WONG 81]. It is assumed that a look-up table will eventually be available at each site [HEVN 79]. During the normal operation of the system, a certain user query issued at some site will always look for a specific CRDU at a specific site. The system in charge of selecting the most cost beneficial copies of data to be included in a query materialization is called the "integrity subsystem" in [HEVN 79]. The next issue is query analysis. It is mainly concerned with the decomposition of a query statement, expressed in a high level language, into simpler subqueries [WONG 77]. This leads to the accomplishment of elementary processing tasks.

Query optimization is the final issue of query processing and it is the objective of the "query processing subsystem" [CCA 80, HEVN 79]. To visualize the optimization of a distributed query the relational algebra is now mentioned. The relational algebra is basically defined through its operations on relations, mainly, *Selection, Projection* and *Join* [CCA 80]. Each data processing task can be mapped to one of those algebra operations. A join can be substituted by a pair of *semi-joins*. In most practical cases this substitution actually results in less communication overhead [CCA 80, CHUN 83]. Whereas the joining of two relations necessitates the displacement of at least one whole relation to a common site, a semi-join operation can be performed by keeping the relations at their local sites, and by transmitting their respective joining columns to each other. Hence, joining several relations, which incidently is a commutative and associative operation [AHO 77], can be achieved by a sequence of semi-joins. This implies that one semi-join can take advantage of another in terms of data-volume [CCA 80]. At this point it is worthwhile to pause and make the following remark. One query may be solved by more than one sequence of such selections, projections and semi-joins. It is commonly accepted that all the restrictions and projections must be performed locally, as far as possible, to reduce the size of the initial relations. Those smaller-sized relations are referred to as *fragments*. It is assumed that an efficient sequence of semi-joins can be found prior to addressing the allocation problem explicitly. That it is correct to do so results from the fact that, in a packet switching network, delays are mainly proportional to the flow of data-volume, and therefore, in such an environment, any optimal query processing sequence depends mainly on the partitioning of accessed relations, and not so much on their positions in the network [CHUN 83].

The quality of a processing sequence is measured by the cost of its semi-joins (the volume of data to be transmitted) and their benefits (the volume of data eliminated by a semi-join operation [CCA 80, CHUN 83]). Algorithm *OutTree* of Section 4.4 eliminates, as much as possible, the need to access attributes that do not belong to the target list of a query, while minimizing some measure of cost associated with semi-joins.

No distributed query processing strategy is, as of now, standard. However, the SDD-1 algorithm [CCA 80] contains all the features of a realistic strategy. "Algorithm H" of [CHUN 83] follows the same spirit proposed in [CCA 80], but goes further towards the optimization of semi-join procedures. During a distributed query processing session, a query originates *file-to-file* communications to execute some data-reducing operations, or simply, *reducing operations*. Once those operations are completed, *file-to-user* communications take place to collect the result, at the originating user site. This action of collecting the final relevant data is referred to as the *assembly operation**. It is important to note that the flow incurred by file-to-file transmissions as well as the assembly flow are much larger than the flow corresponding to messages due to requests and control [CHUN 83]. The reducing flow is based upon the transmission of large volumes of data, sometimes complete columns of a relational table at a time, associated with join or semi-join [CCA 80] operations. For updates however, transmissions are mainly due to messages. However, transmissions of messages associated with concurrency control mechanisms are not considered, neither are the time delays due to transmission and waiting in queues [MAHM 76, KHAB 80]. The main focus is on the minimization of

*Note that the terms *assembly* and *reducing* were previously used in connection with the flow rather than the logical operations that lead to that flow. Such a flexibility in terminology will be continued.

data-volume shipments.

In the final section, some characteristics of earlier data allocation models are surveyed. Modeling assumptions and mathematical aspects are discussed.

2.5. Data allocation models

Early research on data allocation has been reported in [CASE 72, CHU 73, ESWA 74, MAHM 76, MORG 77]. Further studies related to the same problem include [KHAB 80, FISH 80, RAMA 79b, APER 80]. In all the publications mentioned above, the assumption of file independence is made. This means that, given any query, the complete answer to the query is the collection of the partial answers contained in all the relevant files accessed. Unfortunately, this assumption is entirely inadequate when applied to a distributed database that is represented by a set of relational tables. As was noted above, distributed query processing does not consist in a mere file-to-user access, but incurs file-to-file communication, as well, between sites other than the user site [BOOT 81, CCA 80]. When considering a network of autonomous computers [TANE 81], and users at each site, with roughly the same needs in terms of logical accesses, it is hazardous to claim the feasibility of designing independent files that will seldom necessitate any file-to-file communication when processing queries. Relations are not independent, they are related by the attributes they have in common, and therefore, they must be allowed to participate in cross-referencing operations.

A model for the placement of relations in a network, that apparently departs from classical ones is presented in [RAMA 79b]; it applies to a relational database in which cross-referencing accesses are permitted. However, the query processing policy assumed in [RAMA 79b] requires that the relations accessed in a multi-relation retrieval, must be moved simultaneously to the originated site before retrieval operations can be performed. In essence, this amounts to restricting the query processing communications to file-to-user communications, and therefore the flow of data associated with the reducing operations, which has been referred to as the file-to-file flow, is not considered.

In [APER 80], however, the problem of distributing a set of relations in a computer network is considered from a different viewpoint. Indeed, the author acknowledges the importance of including cross-referencing in the distribution stage. An efficient heuristic is used to allocate already existing relations to the sites of the network. The two last mentioned studies do not address some very typical issues in the distribution of CRDUs; these are introduced below.

There is one main difference between distributing, on one hand, files which do not feature the cross-referencing property, and on the other hand, CRDUs, and in particular relations, which by definition can participate in cross-referencing accesses with one another because of semantic dependence. In the first case it is acceptable to evaluate the total data-volume flow using the superposition principle, whereas in the second case the data-volume contributions are ruled by highly nonlinear effects due to *clustering*. To understand the significance, on data-volume flow, of clustering CRDUs, let us consider Figure 2.1. It depicts two possible cases of relation clustering for a query, originated at site t, which needs to access relations R1[A,B], R2[A,C] and R3[A,D]. In Figure 2.1-i R1 and R3 are clustered* at site 1. R2 is available, by itself, at site 2. The query may be processed by following a strategy as depicted by the graph of Figure 2.1-i. On this directed graph, arc (a,b) refers to a "dummy"** move of relation R3 towards R1 resulting in a join R1[A]R3. Also, each vertical arc such as (b,e) indicates that no data flow is associated with the corresponding access of data because no physical inter-site move is required. Arc (b,d) represents a reducing operation from R1[A]R3 towards R2 initiated at step 0 of the distributed query processing sequence; it corresponds to the semi-join $(R1^0[A]R3^0)[A{>}R2^0$ which requires the shipment of a volume of data labeled $(R1^0[A]R3^0)[A]$ † . Arc (d,e) corresponds to a reducing operation from R2

*We may visualize the clustering of R1 and R3 as resulting from the *join* of R1 with R3 over column A if it is advantageous to do so.

**It obviously incurs no flow since R1 and R3 are located at the same site.

†The superscripts indicate the step in the distributed query processing sequence. R1[A]R3 sends all the values of A that it possesses, to site 2 where R2 is located. We assume familiarity with the usual notation for *join*, *semi-*

towards R1[A]R3, initiated at step 1; it is labeled $R2^1[A]$ to indicate that the copy of R2, at step 1, sends its own column of values for the attribute A, to site 1. The assembly flow results from a transfer of relations $(R1^0[A]R3^0)<A]R2^1$ and $R2^1$, respectively from sites 1 and 2, to the user site t. For the case depicted in Figure 2.1-ii, R1 and R2 are clustered at site 1. They are represented by their join R1[A]R2 at site 1, whereas R3 is available at site 2. The flow of data-volume resulting from this configuration is indicated in Figure 2.1-ii. It is assumed that the query processing strategy depicted seeks to minimize the flow of data-volume for this new configuration of the initially accessed relations. The distributed query processing strategy of Figure 2.1-ii is shown to be totally different from the previous one in Figure 2.1-i. The reason is that in each case, the particular representation of the initially accessed relations is fully taken advantage of by the distributed query processing algorithm used [CHUN 83]. Therefore the two configurations of R1, R2 and R3 in the network, result in different combined file-to-file and file-to-user flows. Obviously, the *clustering* of CRDUs affects in a non-trivial way the flow of data-volume associated with the distributed query processing mechanism.

Yet another problem arises from overlooking the importance of the position, in the network, of the user who originates a query. Figure 2.2-i shows an optimal query processing strategy to solve a query by operating on three distinct CRDUs r, q1 and q2. It is assumed that those CRDUs will be placed at different sites, and hence the clustering effect needs not be considered. Figure 2.2-ii displays the reducing and assembly flows in the network when q1 is at site i, q2 at site j, r at site k and the user site t is distinct from i, j and k. If the user is moved to site k, then site t becomes identical to k; the flow would look as shown in Figure 2.2-iii. The assembly flow resulting from the shipment of the processed copy of r to site t is discarded. However, one question still remains; looking at Figure 2.2-i, it can be noticed that arcs [(i,2),(t,3)] and [(j,2),(t,3)] correspond to cross-referencing operations aiming at reducing the size of the copy of CRDU r, and therefore the assembly flow from k to t. Furthermore, because no other cross-referencing operation is to either benefit or follow from their

join and *projection*.

application, it is said that these cross-referencing operations are *noncritical*. Of course it seems futile to try to decrease the size of r because this will not trigger any move to decrease the sizes of q1 or q2 any further, and because r is already at the destination site t! Therefore the arcs which are represented with dashed lines can be omitted from Figure 2.2-iii, illustrating the fact that, even without the effect of clustering, the total flow, in processing a query, may be affected by the location of the site at which the query is originated. The term *non-critical* will be used in association with a cross-referencing operation or a semi-join, in the context of distributed query processing when discussing the optimization model. A more formal definition will be provided in the next chapter.

The next remark is about the *materialization* issue. Figure 2.3 depicts a user, at site t, who originates a query that may be answered through the access and cross-referencing of CRDUs R1, R2 and R3. The figure depicts a multiple copy situation. The query needs only access a *non-redundant instance* of the set {R1, R2, R3}; this is referred to as a *static materialization**. Such materializations are not known in advance, therefore, at the design stage, the *materialization* problem must be addressed jointly with the *distribution* problem.

Finally, stringent physical constraints may have to be considered at the design stage. This is particularly true if the supporting computer network of the distributed system does not include communication links which feature virtually infinite flow capacity characteristics. For instance, if *static routing* is used to direct the traffic in the network, it may not be practical to always follow the shortest path from a node i to a node j whenever some communication is to take place between node i and node j. Consequently, any isolated communication between i and j may be associated with a set of superposing paths from i to j. If *adaptive routing* techniques are used, the flow is forced to follow the least busy routes in the network. At the design stage, for both routing techniques mentioned, finite flow capacity may affect the distribution of the CRDUs. This issue is most easily dealt with when a descriptive

*As opposed to a *dynamic* materialization [WONG 81], which may include redundant copies and which may allow the query processing to take advantage of parallel processing of those redundant copies.

mathematical model is available.

The computational methods applied to allocation problems are usually based on integer or mixed-integer programming algorithms. A few of them are surveyed below*. The scope of the problem analyzed in [MAHM 76] is larger than those of most file allocation formulations. It includes a wider description of the computer network, such as links and nodes *reliabilities,* and file *availability.* Time delays are also considered explicitly. The final model is a nonlinear integer program. A nonlinear branch-and-bound integer programming technique is adopted for its solution; it turns out to be unsuccessful for problems that are not of small size. In [KHAB 80], the file allocation problem is combined with the communication network design (i.e. its topology and channel capacities assignment). The communication cost is identified by a non-linear integer (0-1) expression because the flow is estimated in terms of the zero-one allocation variables and probabilities associated with the access policy. Furthermore, the number of copies for each file are fixed by the file availability requirements. Hence, the proposed design approach leads to a simple form of the allocation model. The file allocation routine is based upon a greedy algorithm that has been tested with as many as ten sites. In [MORG 77], the problem includes the allocation of programs as well as files. The associated mathematical model is a mixed-integer linear program. In the solution of this model, each possible file assignment is represented by a vertex on an n-dimensional hypercube. A partial search is initiated with any feasible file assignment. A sequence of file assignments are generated by branching forward on the hypercube. Through the execution of the algorithm, a feasibility test is consulted to determine wether some portions of the hypercube can be discarded. The model described in [FISH 80] is based on simple file-to-user accesses and on the premise that the whole database can be represented by one file. Overall it features the most assumptions in terms of database and network requirements. Nevertheless, the problem leads to a mixed integer program as in [MORG 77], and the algorithm presented is valuable in its own right regardless of the initial assumptions. The algorithm is based on heuristic building blocks for

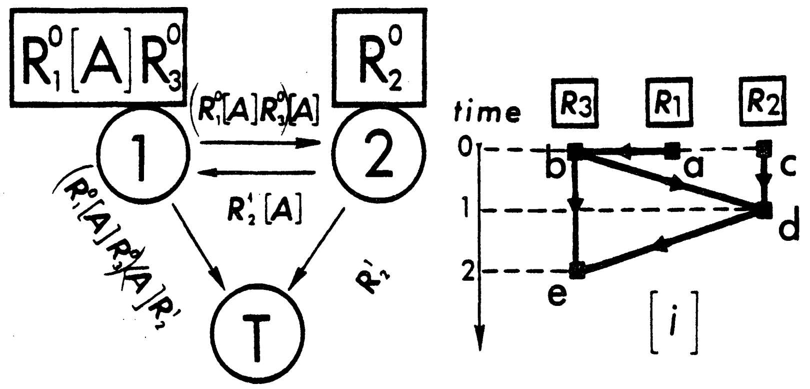
*None of them deals with the issue of cross-referencing in distributed query processing.

generating feasible solutions. A lagrange relaxation approach is followed for obtaining lower bounds on the cost of an optimal solution. [FISH 80] also includes the development of a dominance test that can be used in a preprocessing step to reduce the size of the problem before submitting it to a branch-and-bound algorithm. This algorithm has been shown to withstand a relatively large scale testing. It has been proved in [ESWA 74] that the classical file allocation problem is np-complete. Since then, researchers have encouraged the use of heuristics to solve such problems [CHAN 77, RAMA 79a]. Also, some possible pragmatic techniques applicable to data allocation have been described in [BUCK 79, HAMM 79, PALM 79].

In the following chapter (Section 3.1) a formal definition for some of the technical terms encountered so far* is given. The problem of interest is then worded in a more precise way (Section 3.2) and a solution procedure is sketched (Section 3.3).

*Other technical terms are briefly defined in Appendix A in casual words. For more complete definitions, the references should be consulted.

←	**Local join: No inter-site move. No delay.**
↓	**Participation in a semi-join: No inter-site move. Finite delay.**
↘ ↙	**Participation in a semi-join: Inter-site move. Finite delay.**
↘↓	**Semi-join.**



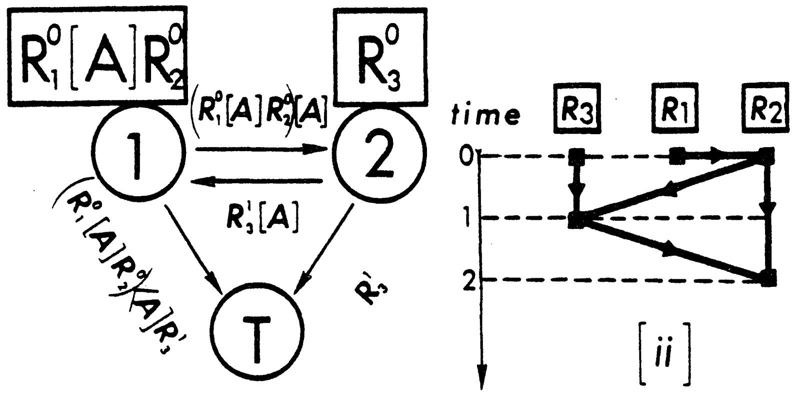


Figure 2.1. The effect of data clustering on distributed query processing

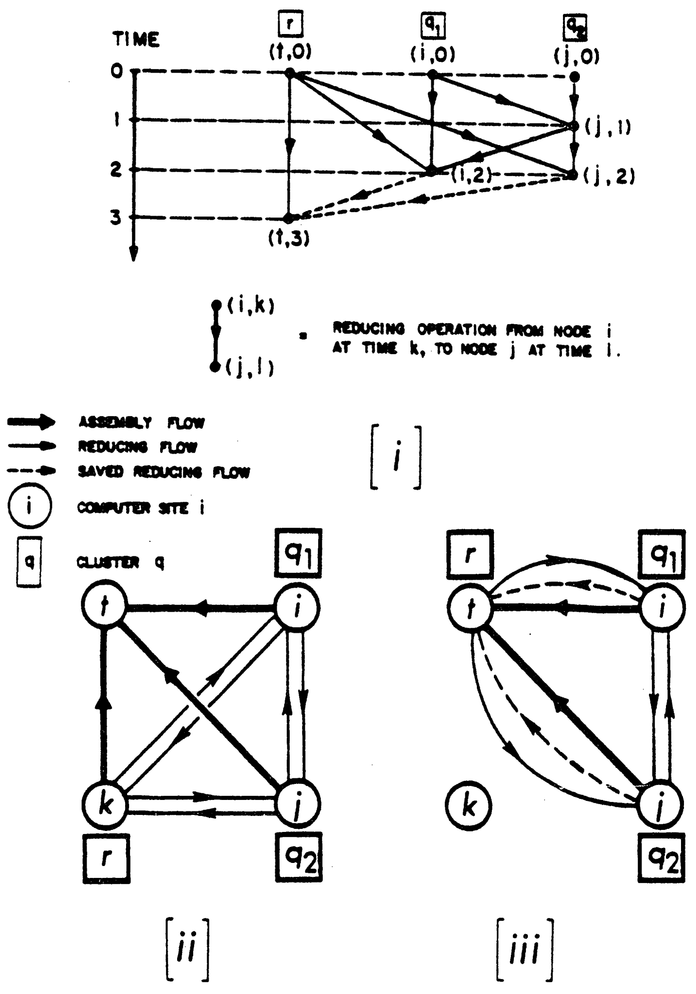


Figure 2.2. Noncritical semi-joins in distributed query processing.

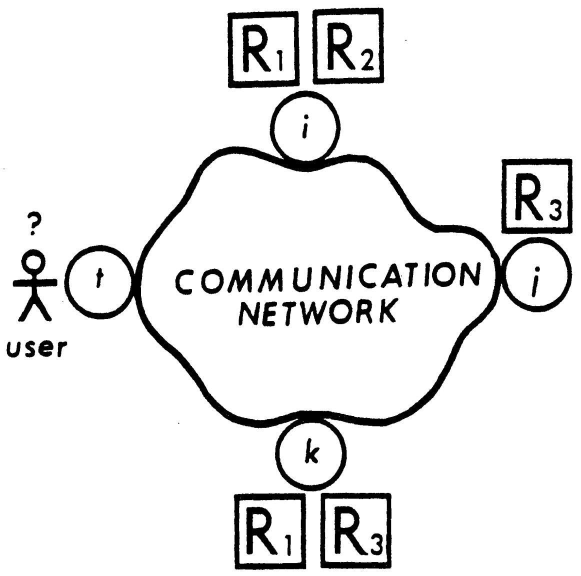


Figure 2.3. The materialization problem

CHAPTER 3

PRELIMINARY FRAMEWORK

This chapter is divided into three sections. Section 3.1 contains basic definitions with appropriate references. Section 3.2 provides a concise statement of the problem of interest in more formal terms. Section 3.3 proposes general guidelines to tackle the problem.

3.1. Definitions

In this section definitions of the terms *Functional dependency* (FD), *closure* and *covers* of a set of FDs, *relation scheme,* and *database schema* are presented.

3.1.1. Functional dependency and database schema

Let U be a *universe of attributes.* Every attribute a_i in U takes its values from a *domain* set $DOM(a_i)$. Let $X = \{a_1, \cdots, a_n\}$ and $Y = \{b_1, \cdots, b_m\}$, ($X \subseteq U$ and $Y \subseteq U$) be ordered sets of attributes. Suppose there exists a function f

$$f : DOM(a_1) \times DOM(a_2) \times \cdots \times DOM(a_n) \rightarrow DOM(b_1) \times \cdots \times DOM(b_m),$$

i.e. for any $x, \bar{x} \in DOM(a_1) \times \cdots \times DOM(a_n)$, at any time, if $x = \bar{x}$, then $f(x) = f(\bar{x})$. A *functional dependency* (FD) is said to *hold* from X to Y, denoted by $X \rightarrow Y$, if there exists such a function f. More casually, f is said to hold on $X \bigcup Y$. We also say that X *determines* Y. $X \bigcup Y$ is called the *attribute scheme* of f and is denoted by $S(f)$. If both $X \rightarrow Y$ and $Y \rightarrow X$ hold, we write $X \longleftrightarrow Y$. We will use capital sized letters from the end of the alphabet, such as X, Y, Z, to denote sets of attributes. Single attributes will be denoted with small case letters from the top of the alphabet, such as a, b, $\cdots$

FD inference rules, known as *Armstrong's rules,* have been studied in the literature [ARMS 74]. These are:

FD inference rules

FD 1 (Reflexivity) $Y \subseteq X$, yields $X \rightarrow Y$,
FD 2 (Augmentation) $Z \subseteq W$ and $X \rightarrow Y$
    yield $XW \rightarrow YZ$ (where XW stands for $X \bigcup W$),
FD 3 (Transitivity) $X \rightarrow Y$ and $Y \rightarrow Z$ yield $X \rightarrow Z$,
FD 4 (Pseudotransitivity) $X \rightarrow Y$ and $YW \rightarrow Z$
    yield $XW \rightarrow Z$,
FD 5 (Union) $X \rightarrow Y$ and $X \rightarrow Z$ yield $X \rightarrow YZ$,
FD 6 (Decomposition) $X \rightarrow YZ$ yields $X \rightarrow Y$ and $X \rightarrow Z$.

A set of inference rules is said to be *complete* if it implies rules FD 1-6. Relaxed versions of FD 1 and FD 2, together with FD 4, can be shown to constitute a complete set of inference rules [KAMB 78]. These are:

(i) $X \rightarrow X$,
(ii) $X \rightarrow Y$ yields $XW \rightarrow Y$,
(iii) $X \rightarrow Y$ and $YW \rightarrow Z$ yield $XW \rightarrow Z$.

The *closure* F^+ of a set of FDs F is the set of all FDs that can be inferred from the FDs in F using a complete set of inference rules. Given some sets of FDs F and G, F is a *cover* of G if $F^+ = G^+$.

A set of FDs F is *nonredundant* if there is no set of FDs G properly contained in F with $G^+ = F^+$ [MAIE 80]. A set of FDs is *minimum* if there is no set G with fewer FDs than F such that $G^+ = F^+$ [MAIE 80]. A set of FDs F is *L-minimum* if:

(1) F is minimum, and

(2) for every FD $X \rightarrow Y$ in F, there is no $\bar{X} \subset X$ with $\bar{X} \rightarrow Y$ in F^+ [MAIE 80].

A set of FDs F is *LR-minimum* if it is L-minimum, and replacing FD $X \rightarrow Y$ in F by $X \rightarrow \overline{Y}$, with $\overline{Y} \subset Y$, alters the closure of F [MAIE 80].

As in [BISK 79], a *relation scheme* is defined to be a pair $<X,E>$ where $X \subseteq U$ and $E \subseteq F^+$ is a set of FDs holding on attributes of X, i.e. if $R \rightarrow S \in E$, then $R \bigcup S \subseteq X$. Note, however, that even though $R \rightarrow S$ may hold on $R \bigcup S$, $<R \bigcup S, \emptyset>$ is a legitimate relation scheme, where $\emptyset$ denotes the empty set. If only the first entry or the second entry of the pair $<X,E>$ is relevant, we use the notation $<X,>$ and $<,E>$ respectively. The relation scheme $<U,F>$ is called the *universal relation scheme.*

A *database schema* is a set $D = \{<X_1,F_1>, \cdots ,<X_n,F_n>\}$ where $\bigcup_{i=1}^{n} X_i = U$ and $(\bigcup_{i=1}^{n} F_i)^+ = F^+$, and the FDs of F_i hold on X_i. Note that this definition is more restrictive than the one in [BISK 79]. A *database subschema* is a set $\overline{D} = \{<Y_1,E_1>, \cdots ,<Y_m,E_m>\}$ where, (1) the set Z defined as $Z \overset{\Delta}{=} \bigcup_{i=1}^{m} Y_i$ satisfies $Z \subseteq U$, (2) the set E defined as $E \overset{\Delta}{=} \bigcup_{i=1}^{m} E_i$ satisfies $E^+ \subseteq F^+$, (3) the FDs of E_i hold on Y_i.

For $X = \{a_1, \cdots ,a_n\} \subseteq U$ (and X is ordered) a finite set $r \subseteq DOM(X) = DOM(a_1) \times \cdots \times DOM(a_n)$ is defined to be a *relation* on X. We also denote such a relation by $r[X]$. Each element t of r is a *tuple* of values defined on X. (A tuple t can be seen as a function from a set of attributes X to $DOM(X)$). An FD $Y \rightarrow Z$ is said to hold in a relation r defined on X if it holds on some subset of X, i.e. $Y \bigcup Z \subseteq X$. Note that for any two tuples t and s in r, if $t(Y) = s(Y)$ then $t(Z) = s(Z)$.

A class of *instances* described by the relation scheme $<X,E>$ is $I_{<X,E>} = \{r : r$ is a relation on X, and all FDs of E hold in $r\}$. An element of $I_{<X,E>}$ is called a *relation instance*. A database schema $D = \{<X_1,F_1>, \cdots , <X_n,F_n>\}$ describes the following class of *instance sets:*

$$I_D = \{\{r_1, \cdots, r_n\} : r_i \in I_{<X_i,F_i>} \text{ for } i = 1, \cdots, n\}.$$

If $\bar{X} \subseteq X$ then $t(\bar{X})$ denotes the subtuple containing the $|\bar{X}|$ components of $t(X)$ corresponding to the elements of $\bar{X}$. The *projection,* over $Y \subset X$, of a relation r defined on X is $r[Y] = \{\bar{t} : t \in r \text{ and } \bar{t} = t(Y)\}$. The *natural join* of two relations, $r[X]$ and $s[Y]$, denoted by $r[X \cap Y]s$, or simply $r[]s$, is $r[X \cap Y]s = \{t : t(X) \in r[X] \text{ and } t(Y) \in s[Y]\}$. The *semi-join* of relation $s[Y]$ with relation $r[X]$, denoted by $r[X \cap Y\!>s$ or simply $r[>s$, is $r[>s = (r[]s)[Y]$.

A database schema $D = \{<X_1,>, \cdots, <X_n,>\}$ is said to feature (or satisfy) the *lossless join property,* or display a *join dependency,* with respect to $<U,F>$ if, for all $r \in I_{<U,F>}$, $r = \underset{i=1}{\overset{n}{[]}} r[X_i]$, where the notation on the right hand side of the equality means that all relations are joined together. This notation is appropriate because the join operation is associative and commutative. Let $\bar{D}$ be a subschema $\{<Y_1,E_1>, \cdots, <Y_m,E_m>\}$ and let $Z \overset{\Delta}{=} \bigcup_{i=1}^{m} Y_i$ and $E \overset{\Delta}{=} \bigcup_{i=1}^{m} E_i$. $\bar{D}$ is said to satisfy the lossless join property with respect to $<Z,E>$ if for all $r \in I_{<Z,E>}$, $r = \underset{i=1}{\overset{m}{[]}} r[Y_i]$.

The concept of *universal relation scheme* or simply, *universal relation* (UR), has been discussed by many researchers; see for instance [BEER 81, ULLM 82, FAGI 82, MAIE 83]. The implications of such a concept are controversial. However, there is some agreement that the universal relation is both useful and practical [MAIE 83]. Furthermore, to quote [BEER 81]:

In almost any "real world" situation, a single join dependency suffices, together with some functional dependencies, to define the legal relations that might be the universal relation at some time.

In order to render the concept of *universal relation* acceptable the following assumptions are made:

Assumption UR-1: *If an FD holds from X to Y, then it is unique. In particular, $X \rightarrow X$ denotes the identity FD over X.*

Assumption UR-2: *The attributes are named in such a way that rule FD 8 is satisfied.*

Assumptions UR-1 and UR-2 imply that sufficient renaming of attributes needs to be made.

Assumption UR-3: *Any database schema D satisfies the lossless join property with respect to $<U,F>$.*

Assumptions UR-1 and UR-3 are discussed in [ULLM 82], and Assumption UR-2 is necessary if problems pointed out in [ATZE 82] are to be avoided.

3.1.2. F-graphs

In this section, a graphical means of depicting the FDs of the universal relation scheme $<U,F>$, first used in [LOZI 78], is introduced.

Let $<U,F>$ be a universal relation scheme where $F = \{f_1, \cdots, f_n\}$. Let $G = <N_G, A_G>$ be a bipartite graph where $N_G = V \bigcup W$ for some V and W, $A_G = A_v \bigcup A_w$ for some A_v and A_w, and $V \bigcap W = \emptyset$, $A_v \bigcap A_w = \emptyset$. G is said to be the *F-graph* of $<U,F>$ if and only if:

For all $f_i \in F$ where f_i stands for $X_i \rightarrow Y_i$ (for some X_i , $Y_i \subseteq U$) there exist,

(1) $V^i = \{v_j : a_j \in (X_i \bigcup Y_i)\}$ where v_j denotes the only node which is labelled a_j,

(2) $w^i \in W$ is unlabelled and only associated with f_i,

(3) $A_v^i = \{(v_k, w_i) : a_k \in X_i\}$,

(4) $A_w^i = \{(w_i, v_m) : a_m \in Y_i\}$,

and further:

(1) $V = \bigcup_{i=1}^{n} V^i$,

(2) $W = \{w_1, w_2, \cdots, w_n\}$,

(3) $A_v = \bigcup_{i=1}^{n} A_v^i$,

(4) $A_w = \bigcup_{i=1}^{n} A_w^i$.

Any FD $f_i \in F$ is said to be *explicitly represented* (or simply, *explicit*) in G. V is called the set of v-nodes, and W the set of w-nodes. Given a node x, the set of *predecessors* of x, $\pi(x)$, is $\pi(x) = \{y : (y,x) \in A_G\}$ and the set of *successors* of x, $\sigma(x)$, is $\sigma(x) = \{y : (x,y) \in A_G\}$.

A v-node y is *reachable* [LOZI 80] from a set of v-nodes $\overline{V} \subseteq V$ if:

(1) $y \in \overline{V}$, or

(2) There exists $w_j \in W$ and $(w_j, y) \in A_w$ and all $\overline{v} \in \pi(w_j)$ are reachable from $\overline{V}$.

A w-node y is reachable from a set of v-nodes $\overline{V} \subseteq V$ if all the nodes of $\pi(y)$ are reachable from $\overline{V}$. A set of v-nodes $\overline{V}$ is said to *lead* to another node y if y is reachable from $\overline{V}$. A w-node x of an F-graph G leads to another node y of G if y is reachable from $\pi(x)$. A set of w-nodes $\overline{W}$ leads to another node y if all $x \in \overline{W}$ lead to y. In general, a set of nodes X leads to a set of nodes Y (we write $X > Y$) if X leads to every node of Y. A *source* of a set of v-nodes $\overline{V}$ is defined to be any w-node w_s that leads to $\overline{V}$.

If v_i is a v-node and a_i its attribute label, the a_i is also denoted by $Att(v_i)$. If $\overline{V} = \{v_1, v_2, \cdots, v_n\}$ is a set of v-nodes, $\{Att(v_i) : v_i \in \overline{V}\}$ is denoted by $Att(\overline{V})$.

Given a w-node w_s and a set of v-nodes $\overline{V}$ such that $w_s > \overline{V}$, $Q \subseteq U$ is a *key* of $Att(\overline{V})$ if and only if:

(1) $Q = Att[\pi(w_s)]$,

(2) for any $\overline{w}_s : \overline{w}_s > \overline{V}$, $\pi(\overline{w}_s) \not\subset \pi(w_s)$.

If a set of FDs F is LR-minimum, then the corresponding F-graph is said to be *minimum*.

An F-graph is said to be *lossless* if the database schema defined by $D = \bigcup_{f_i \in F} \{<,\{f_i\}>\}$ satisfies the lossless join property with respect to $<U,F>$..

3.1.3. Link and envelope

Defined below are, a *link* (we will compare our definition with that of [LOZI 80]) from a set of w-nodes to a set of v-nodes, and an *envelope* for a set of v-nodes.

Suppose there exists a sub-graph $\bar{L}(w_s,v) = (V_{\bar{L}} \bigcup W_{\bar{L}}\ ,\ A_{V_{\bar{L}}} \bigcup A_{W_{\bar{L}}})$ of G where $w_s \in W$ and a $v \in V$ and $w_s > v$ in $\bar{L}$, but for any $y \in V_L \bigcup W_L$ $w_s \not> v$ in $\bar{L} - \{y\}$, and further, if any arc e is deleted from $A_{V_{\bar{L}}} \bigcup A_{W_{\bar{L}}}$, then $w_s \not> v$ in $\bar{L} - \{e\}$, then $\bar{L}(w_s,v)$ is called a *proper link* from w_s to v. We say that any arc in $\bar{L}$ is a *linking arc* from w_s to v with respect to $A_{V_{\bar{L}}} \bigcup A_{W_{\bar{L}}}$. w_s and v are respectively called the *source* and the *sink* of $\bar{L}$. If $\bar{L}^1(w_s,v) \cdots \bar{L}^{n_v}(w_s,v)$ stand for all the proper links from w_s to v, then $L(w_s,v) \overset{\Delta}{=} \bigcup_{i=1}^{n_v} \bar{L}^i(w_s,v)$ is simply called the *link* from w_s to v. The *link* $L(w_s,T)$ from w_s to $T \subseteq V$ is defined by: $L(w_s,T) = \bigcup_{v \in T} L(w_s,v)$. An F-graph $\bar{L}(w_s,T)$ defined by $\bar{L}(w_s,T) = \bigcup_{\substack{v \in T \\ i_v \in \{1, \quad ,n_v\}}} \bar{L}^{i_v}(w_s,v)$ is called a *sub-link* of L (or simply, a sub-link). If for any node y in $\bar{L}$ we have that $w_s \not> T$ in $\bar{L} - \{y\}$, then $\bar{L}$ is said to be proper. Given $W_s \subseteq W$, the link from W_s to T is $L(W_s,T) = \bigcup_{w \in W_s} L(w,T)$.

An FD $f : X \to a_i$, is said to be a *base-FD* in an F-graph G if there exists $w \in W$ such that $X = Att(\pi(w))$ and $a_i \in Att(\sigma(w))$.

Given the F-graph G of a universal relation scheme $<U,F>$, an FD $f_i : X \to Y$, $f_i \in F^+$, is said to be induced by a sub-link $L(W_s,T)$ if $X = \bigcup_{v \in V_s} \{Att(v)\}$ where

$V_s \overset{\Delta}{=} \bigcup_{w \in W_s} \pi(w)$ and $Y = \bigcup_{v \in T} \{Att(v)\}$. If such an FD f_i is not explicit in G, it is said to be *implicitly represented* (or simply, *implicit*) in G. The notation $f(w_s, v)$ will denote the FD induced by a proper link $\bar{L}(w_s, v)$.

The *length* of a proper link $\bar{L}(w_s, v_t)$ is defined to be the maximum number of w-nodes, including w_s, encountered on a directed path from w_s to v_t.

If there exist two or more distinct proper links from w_s to v_t, and two of which are $\bar{L}^+(w_s, v_t)$ and $\bar{L}^-(w_s, v_t)$, then the pair $[\bar{L}^+(w_s, v_t); \bar{L}^-(w_s, v_t)]$ is said to be a *doublet*. Let $f^+(w_s, v_t)$ and $f^-(w_s, v_t)$ be the respective representations of $\bar{L}^+$ and $\bar{L}^-$, in the F-graph G. Then, by Assumption UR-1, f^+ and f^- must be identical.

Let $\bar{L}(w_s, v_t)$ be a proper link. Let $\bar{C} \subseteq V_{\bar{L}}$, be such that:

(1) $\bar{C} > v_t$ and,

(2) for any $v_i \in \bar{C}$ $\bar{C} - \{v_i\} \not> v_t$.

Define $C \subseteq W_{\bar{L}}$ as,

$$C \overset{\Delta}{=} \bigcup_{v_i \in \bar{C}} \pi(v_i).$$

A *cutset* Γ of $\bar{L}$ is defined as:

$$\Gamma \overset{\Delta}{=} \{(w_i, v_i) : w_i \in C, \text{and } v_i \in \bar{C}\}.$$

By abuse of notation, we write $\Gamma \equiv (C, \bar{C})$.

Given an F-graph G of a universal relation scheme $<U,F>$, and given a set of v-nodes T, an *envelope* $K(T)$ of T is a subschema

$$K(T) \overset{\Delta}{=} \{<Y_1, E_1>, \cdots, <Y_m, E_m>\} \text{ where,}$$

(1) for any $<Y_i, E_i> \in K(T)$, there exists $f_i \in F^+$ such that $E_i = \{f_i\}$ and $Y_i = S(f_i)$,

(2) for any $v_j \in T$, there exists $<,\{f_i\}> \in K(T)$ such that $Att(v_j) \in S(f_i)$ and,

(3) $K(T)$ satisfies the lossless join property with respect to $< \bigcup_{i=1}^{m} Y_i \ , \ \bigcup_{i=1}^{m} E_i >$.

Such an envelope $K(T)$ is always guaranteed to exist but is not necessarily unique. In the next chapter, we will show that, for any T, for Condition (3) to be satisfied, it suffices to have some set of attribute Y_i be such that $Y_i \rightarrow Att(T)$ holds. In the worst case, Y_i may have to be identical to $Att(T)$. We will also show that given a sub-link $L(w_s, T)$, the set of relation schemes defined as:

$$\bigcup_{(w_j, v_i) \in A_{wL}} \{<S(f(w_j, v_i)) \ , \ \{f(w_j, v_i)\}>\},$$

is an envelope of T. However, we will see that such a sub-link, which has a single source w_s for all its v-nodes, can only be guaranteed to exist in a special kind of lossless F-graphs. We will also show that Assumption UR-3 requires that, for any T, at least one envelope $K(T)$ can be determined for which all the FDs are in F rather than in F^+, as would be the case more generally. Further, that a universal relation scheme $<U,F>$ which satisfies Assumption UR-3 must be depicted with a lossless F-graph.

3.1.4. Distributed query processing sequence

Let $R_1, \cdots, R_I$ denote I relations. Let R represent the list of these relations, i.e. $R = (R_1, \cdots, R_I)$. A *distributed processing sequence* is a sequence R^n, $n = 0, 1, \cdots, K$ such that $R^{n+1} = T^{n+1}[R^n]$, $n = 0, 1, \cdots, K-1$ where T^{n+1} is a transformation that consists of semi-joins executed with respect to some pairs of relation instances of the form (R_i^n, R_j^n) at step $n+1$, and where R^0 is obtained from R after all initial projections and selections have been performed for a given query. The final result of the sequence, R^K, satisfies $R^K = \prod_i(R_i^K) = \prod_i(R_i^0)$. Any semi-join $R_i^{K-1} [> R_j^{K-1}$ contributing to the transformation T^K is said to be an *noncritical* semi-join with respect to the sequence. All the other semi-joins are said to be *critical*. In Figure 2-i (Chapter 2), a distributed query processing sequence was depicted using a directed graph.

3.2. Initial problem formulation

An informal summary of the problem follows. User provided inputs include: (1) A *universal relation scheme* which consists of (i) a universe of *attributes,* (ii) a set of *functional dependencies* (FD) holding on the attributes, and (iii) some statistical information for each attribute, the domain from which the attribute gets its values, and for each FD, (2) a computer network with tightly or loosely connected nodes. It is assumed that the communication network has a fixed topology and that there is enough over-all channel capacity. Communication costs, and channel capacities are assumed to be given. (3) The user's scheduled queries and updates. A query is assumed to consist of a *target list* of attributes, a usage frequency, and a set of *qualification clauses* holding on attributes. An update is assumed to consist of a target list of attributes, a *type,* i.e. whether the update is a "delete" or an "add", and an occurrence frequency. The first phase of the problem must consist of identifying the FDs which will allow the solution of every query and which will enhance the performance of distributed query and update processing. The second phase of the problem must result with a distribution of FDs in the network which minimizes the operational communication cost.

The problem is expressed in formal terms below. We are given the following: (1) A universal relation scheme $<U,F>$ where U is a set of attributes and F a set of FDs which hold on U (the assumptions UR-1, UR-2 and UR-3 hold on $<U,F>$), and Ω a set of statistical informations for each attribute and each FD**. (2) The graph of a computer network $\Gamma = (N,A)$, where N stands for the set of nodes identified with computer sites, A stands for the edges identified with communication channels. The graph has a fixed topology, and the network has appropriate† channel capacities. Transmission costs characteristics are given. (3) For each user-site $t \in N$, a set of queries is given, defined by

**See Section 4.6.

†This means that there is enough channel capacity overall to accommodate the total traffic.

$Q_t = \{Q_{tk} \overset{\Delta}{=} <T_{tk}, \nu_{tk}, C_{tk}>, \quad 1 \leq k \leq K_t\}$, where K_t is a positive integer, $T_{tk} = (j_1^{tk}, j_2^{tk}, \cdots, j_{|U|}^{tk})$, and $j_i^{tk} = 1$ indicates that the i^{th} attribute $a_i \in U$ is referenced in the query, $= 0$ otherwise. ν_{tk} is the frequency of occurrence of query Q_{tk}, and C_{tk} is a set of qualification clauses specified with respect to the universal scheme $<U, >$. For each user-site $s \in N$, a set of updates is given, defined by $U_s = \{U_{sk} \overset{\Delta}{=} <T_{sk}, \nu_{sk}, TYPE>$, $1 \leq k \leq \overline{K}_s\}$, where $\overline{K}_s$ is a positive integer, T_{sk} is defined as T_{tk} with the subscript s substituted for the subscript t, and $j_i^{sk} = 1$ indicates that the i^{th} attribute $a_i \in U$ is referenced in the update. ν_{sk} is the frequency of occurrence of U_{sk}, and *TYPE* is either "ADD" or "DELETE". (A query (Q_{tk}) or an update (U_{sk}) is called a transaction. Given a transaction Q_{tk} or U_{tk}, the set $T = \{a_i : a_i \in U \text{ and } j_i^{tk} = 1\}$ is called the target list of the transaction. The corresponding set of v-nodes in the F-graph is referred to as the set of *target nodes.)* The following is to be achieved: (1) For each query Q_{tk} find an envelope $K_{tk} \subseteq \{<,\{h_i\}> : h_i \in F^+\}$ whose FDs contain the attributes referenced in Q_{tk}. (2) Distribute on Γ the CRDUs mapped to $\bigcup_{Q_{tk}} K_{tk}$ so as to minimize the total communication cost.

3.3. Procedure towards a solution

The task of organizing CRDUs in the computer network is divided into two parts: (1) The determination of *envelopes* for all queries, i.e. the selection of appropriate FDs, and (2) the distribution of CRDUs mapped to the selected FDs.

The main steps according to which the distributed database design problem is addressed are outlined:

PART 1: **Selection: From users' FDs to envelopes**

Given an initial set of FDs, F, a universe of attributes U, and a set of transactions for each user node, the following is to be achieved: For each query, one envelope which leads to the efficient processing for that particular query is to be determined. The following approach is proposed*:

‡Step-1: Given an F-graph, build a minimum F-graph G (Algorithm *Reduce*). The full reduction of the initial user-given F-graph leads to extensive decrease in processing effort later on in the design method. It is shown that Algorithm *Reduce* is slightly less complex than [MAIE 80]'s algorithm. Given an F-graph, G, make minor modifications to satisfy losslessness, as required by Assumption UR-3. This step ensures that a query referencing any set of attributes can be solved correctly. Algorithm *NoLoss* addresses the problem as a simple covering problem.

†Step-2: Given a set of target nodes (for every query), find the sets of w-nodes from which the target nodes are reachable (Algorithm *GetSources* [LOZI 80]); these w-nodes are referred to as *source nodes*. Find the nearest set(s) of source nodes (Algorithm *GetNearestSource* [LOZI 80]). Find the link from the source nodes to the target nodes. This is accomplished by Algorithm *GetLink* which is a corrected version of *Algorithm 2* in [LOZI 80]. This step consists of extracting, from the F-graph, the subgraph which is sufficient to solve the query under consideration.

‡Step-3: For each query, given a link from a set of source nodes to a set of target nodes, extract FDs that cover the target list, contain a minimum number of unreferenced attributes, and lead to low-cost distributed query processing. Thus, each query is assigned its own envelope. Each FD, in an envelope, is mapped to a CRDU.

Step-4: Determining input data-volumes in preparation for PART 2. A distributed query processing algorithm such as [CHUN 83]'s may be used.

*Although the approach based on the synthesis of a distributed database through the distribution of CRDUs mapped to FDs is original, some steps of its implementation make use of existing algorithms. For this reason we attach a "double dagger" (‡) to indicate an original contribution, a "dagger" (†) to indicate that modifications have been made to some known algorithms. Other algorithms or techniques are appropriately referenced.

PART 2: **Distribution: Covering the network with CRDUs**

‡Step-5: Given a set of CRDUs, find a minimum cost distribution and materialization.

Step-6: At this stage in the design, for each query, a database subschema that satisfies the lossless join property has been obtained. Since redundant FDs may have been allocated at any one site during Step-5, a nonredundant subset (not just any nonredundant cover) of FDs must be identified at each site. Relations are then synthesized from FDs using [KAMB 78]'s algorithm.

In Chapter 4 the problems of depicting FDs, finding envelopes for target nodes, and selecting FDs that will be mapped to the CRDUs in PART 2 are addressed.

CHAPTER 4

DATA MODELING

The first section of this chapter begins with the statement of a known result pertaining to the lossless join property of a set of relation schemes. In Section 4.2, properties of F-graphs are studied and Step-1 of the distributed database design problem is addressed. A set of rules to simplify F-graphs that depict extraneous FDs is then proposed. This problem is related precisely to the problem of finding LR-minimum covers for a set of FDs. Section 4.3 deals with F-graph manipulation algorithms for Step-2. Section 4.4 is devoted to the modeling and solution of Step-3, or the envelope optimization problem. Section 4.5 discusses some important issues of consistency preservation associated with the updating of FDs in a database. Section 4.6 closes this chapter with a complete summary of the data modeling approach, and with illustrative examples.

4.1. Foundation of the distribution approach

It was mentioned, earlier in Chapter 1, that each CRDU, an atomic unit of distribution, is associated with one FD. Supporting arguments for this choice were given. This section consolidates the role of FDs in database distribution. Here, the database distribution philosophy is based on vertical distribution (i.e. with respect to attribute types, exploiting the "projection" operation), and on the synthesis approach [BERN 76, KAMB 78]. The explanation for the latter statement is that, after distribution, the FDs assigned to a site can be used to build a relational database [BERN 76], a network database [HOUS 79], or a hierarchical database [DELO 79]. Horizontal partitioning (i.e. with respect to tuples, exploiting the "selection" operation), is beyond the scope of this thesis, but could still complement the proposed approach without conflict.

The well known result on lossless join [AHO 77, BISK 79, LOZI 80], summarized in Theorem 1, is of central importance to our distribution approach. Given a query with target list Y, and given a database subschema $D = \{<X_1, >, \cdots, <X_n, >\}$ where $Y \subseteq \bigcup_{i=1}^{n} X_i$, Theorem 1 indicates how to guarantee that some relations $r_1, \cdots, r_n$, which are picked from $I_{<X_1,>}, \cdots, I_{<X_n,>}$ respectively, can be joined to solve the query while preserving semantic integrity. Theorem 1 combines two corollaries reported in [BISK 79].

Theorem 1: *Let $D = \{<X_1, H_1>, \cdots, <X_n, H_n>\}$ be a database subschema and $<Z,E>$ a relational scheme such that $\bigcup_{i=1}^{n} X_i = Z$. Furthermore, suppose that H, where $H \overset{\Delta}{=} \bigcup_{i=1}^{n} H_i$, is a covering of E ($H^+ = E^+$) such that for any $R \rightarrow S \in H$, there exists $<X_i, > \in D$ with $R \bigcup S \subseteq X_i$ (condition C). Then there exists $<X_{i_0}, > \in D$ such that $X_{i_0} \rightarrow Z \in E^+$ if and only if D has the lossless join property with respect to $<Z,E>$.*

Proof: Refer to [BISK 79]. •

The next corollary refers to the special case where each relation scheme in the database subschema contains a unique FD. Although Corollary 1 trivially follows from Theorem 1, it provides a criterion which is easier to use than that provided by Theorem 1. This criterion only involves FDs, and thus Corollary 1 suggests that FDs constitute natural units of distribution.

Corollary 1: *Let $D = \{<X_1, \{h_1\}>, \cdots, <X_n, \{h_n\}>\}$ be a database subschema and $<Z,E>$ a relational scheme with $\bigcup_{i} X_i = Z$,* and *$H = \{h_1, \cdots, h_n\}$ such that $H^+ = E^+$, and $X_i = \mathcal{S}(h_i)$ for $1 \leq i \leq n$. Then there exists a $<X_i, \{h_i\}> \in D$ that satisfies*

$X_i \rightarrow Z \in E^+$ if and only if D has the lossless join property with respect to $<Z,E>$.

From that follows a sufficient condition for an F-graph G to be lossless.

Corollary 2: *If an F-graph $G = (V \bigcup W, A_v \bigcup A_w)$ is a sub-link with single source, $G \equiv L(w_s, V)$, then it is lossless.*

The converse is not necessarilly true. As a counter-example, consider the F-graph that depicts two explicit FDs $AB \rightarrow C$, and $A \rightarrow D$.

4.2. Step-1: Initial processing of a user-F-graph

A user-provided F-graph may feature unnecessary redundancy in depicting FDs. Several steps may be taken to simplify the user F-graph while preserving its information-representation content. These steps are presented in Section 4.2.1 in the form of a detailed algorithm, *Reduce*. Section 4.2.2 considers the theoretical implications of Algorithm *Reduce*. Section 4.2.3 indicates how Assumption UR-3 may be enforced on an F-graph.

4.2.1. F-graph reduction

The following propositions state some properties of F-graphs. Typically, if the FD inference rules of Chapter 3 can be applied to some of the explicit FDs of an F-graph, to obtain other explicit FDs, then the latter can be eliminated from the graph, and the graph simplified. The notation X_i will denote a set of attributes, and V_i will denote the corresponding set of v-nodes in an F-graph. If no subscript is used to denote a set of attributes, such as X, then the notation V_X will denote the set of v-nodes associated with X. We will use the fact that, in an F-graph which depicts F, the statement $V_i > V_j$ is equivalent to $X_i \rightarrow X_j \in F^+$ (see [LOZI 80]).

Proposition 1: *Given an F-graph G which depicts F, let $v_T \in V$ be a v-node and $w_T \in \pi(v_T)$. If there exists $V_i \subset \pi(w_T)$, $V_j \subseteq \pi(w_T)$, V_i leads to V_j in G,* and $V_i \bigcap V_j = \emptyset$, *then the arcs in the set* $\{(v_j, w_T): v_j \in V_j\}$ *are extraneous (i.e. do not alter the closure of the FDs explicitly represented in G), and can be deleted from G.*

Proof: Let V_k be defined as $V_k = \pi(w_T) - (V_i \bigcup V_j)$. To show that the arcs in $\{(v_j, w_T) : v_j \in V_j\}$ are extraneous, it suffices to show that X_j is superfluous in the left hand side of the explicit FD $X_i \bigcup X_j \bigcup X_k \rightarrow A_T$ (1). From the hypothesis, $V_i > V_j$, which is equivalent to $X_i \rightarrow X_j \in F^+$. Rule FD-1 says that $X_i \rightarrow X_j \bigcup X_i$ (2) also holds. Using Rule FD-4 on (1) and (2) leads to $X_i \bigcup X_k \rightarrow A_T$. Therefore, A_T is determined by $X_i \bigcup X_k$ without any reference to X_j. Furthermore, since $V_i \bigcap V_j = \emptyset$, the arcs of $\{(v_j, w_T) : v_j \in V_j\}$ are only used to convey the participation of X_j in FD (1), hence they are extraneous. •

Proposition 2: *Given an F-graph G which depicts F, let the FD* $f_i : XY \rightarrow ZS$ *be explicit in G, and let* w_i *be its w-node. If the FD* $X \rightarrow Z$ *can be induced by a link, L, and* $w_i \notin W_L$, *then, the arcs in* $\{(w_i, v_j) : v_j \in V_Z\}$ *are extraneous in G.*

Proof: Since $X \rightarrow Z \in F^+$, by FD 2, the depiction of $XY \rightarrow Z$ is redundant. •

A set of rules are proposed that can be used to simplify an F-graph that depicts extraneous explicit FDs. These reducing rules are based on FD-rules of Section 3.1.1 and exploit Propositions 3 and 4.

A reduced F-graph is an F-graph on which the following rules have been performed.

RULE 1: while CONDITION Ω_0:" *Some w-nodes* $w_1, \cdots, w_n$ *have exactly the same predecessor set,*" **do** REDUCE 1: *Combine all the w-nodes* $w_1, w_2, \cdots, w_n$ *into a unique w-node say,* w_1. *Delete all the arcs incident on* $w_2, w_3, \cdots, w_n$. *Create new arcs* $(w_1, \overline{v}_i)$ *for all* $\overline{v}_i$ *such that* $(w_j, \overline{v}_i) \in A_w$, $2 \leq j \leq n$.
/* RULE 1 makes use of rule FD-5. See Figure 4.1-i. */

RULE 2: while CONDITION Ω_1: "*There exist some* $\overline{V}$ and w *for which* $\overline{V} = \pi(w) \bigcap \sigma(w)$ " **do**

```
    REDUCE 2:
    begin
    for all $\bar{v} \in \bar{V}$, delete $(w,\bar{v})$
    if there is no v-node $v \notin \bar{V}$
    such that $(w,v) \in A_w$, then
      delete $w$ from $W$ and
      delete all the arcs incident on $w$
end
/* RULE 2 uses rule FD-1. See Figure 4.1-iv. */
```

RULE 3: while CONDITION Ω_2: "*An FD $XY \rightarrow Z$ is explicitly represented in the graph* and *$w_i \in \pi(V_Z)$ is its w-node,* and *further $X \rightarrow Z$ is induced by a link $L(W_j, V_Z)$ where $W_j \subseteq \bigcup_{v_i \in V_X} \sigma(v_i)$*" **do**

```
REDUCE 3: /* Makes use of Proposition 2. */
begin
   delete all $(w_i, v_j)$ where $v_j$ is
   a v-node labelled with an attribute in $Z$ $(v_j, \in V_Z)$
   if there is no $(w_i, v_i)$
   such that $v_i \notin V_Z$, then
   for all $v_k \in V$
   and $v_k$ is labelled with attributes of $X$ or $Y$
      delete $(v_k, w_i)$
   delete $w_i$ from $W$
end
/* See Figure 4.1-iii and 4.3. */
```

RULE 4: while CONDITION Ω_3: "*For some $w_T \in W$, there exists a proper subset of $\pi(w_T)$, V_i that leads to another subset of $\pi(w_T)$, V_j* and *$V_i \bigcap V_j = \emptyset$*" **do**

```
REDUCE 4: /* Makes use of Proposition 1 */
begin
   for all $v_j \in V_j$
      delete $(v_j, w_T)$ from $A_v$
end
/* See Figures 4.1-ii and 4.2. */
```

Conditions Ω_0, Ω_1 ,Ω_2 and Ω_3 can easily be implemented; here is how:

(1) Condition Ω_0 :

This condition simply requires that every w-node w_i be checked for the following property: Given w_i , is there a w-node w_j, $w_j \neq w_i$, such that $\pi(w_i) = \pi(w_j)$? This is answered by tracing the F-graph backwards from w_i, and checking if there exists such a w_j. The time complexity for RULE 1 is $O(|W|^2)$.

RULE 1:

```
for all $w_i \in W$ do
begin
   for all $w_j \in W - \{w_i\}$ do
   begin
      if $\pi(w_i) = Predecessor(w_j)$
      then  REDUCE 1
   end
end
```

(2) Condition Ω_1:

This condition is the most easily verified. For each w-node w, $\pi(w)$ and $\sigma(w)$ are compared. RULE 2 has a time complexity $O(|W|)$.

RULE 2:

```
for all $w \in W$ do
begin
   if $\pi(w) \subseteq \sigma(w)$
   then  REDUCE 2
end
```

(3) Condition Ω_2:

For every w_i, this condition may be checked by trying to find a link $L(W_j, V_Z)$ which does not contain w_i, where $\bigcup_w \{\pi(w) : w \in W_j\}$ is contained in $\pi(w_i)$, and $V_Z \subseteq \sigma(w_i)$. This is accomplished by starting from $\pi(w_i)$ and by traversing the F-graph forwards in search of such a link (ALGORITHM *ReachAll*, Sect. 4.3, of time complexity $O(|A_v| + |A_w|)$, can be used). Note that we do not need to identify V_X (defined in the statement of CONDITION Ω_2) explicitly. All we need to check is if such a V_X exists at all. The complexity of RULE 3 is $O(|W| \times (|A_v| + |A_w|))$.

RULE 3:

```
for all $w_i \in W$ do
begin
```

```
    Look for a set, of v-nodes , S, reachable (not trivially)
    from $\pi(w_i)$ without using $w_i$.
    (ALGORITHM ReachAll Sect.4.3)
    $V_Z := S \cap \sigma(w_i)$
    if $V_Z \neq \emptyset$
    then REDUCE 3
end
```

(4) Conditions Ω_3:

Condition Ω_3 is slightly less obvious than the others. The reason is that one must try to look for a link (of any length) from one subset of a set of v-nodes, P, to another subset of P. (Algorithm *ReachAll,* Sect. 4.3, of complexity $O(|A_v| + |A_w|)$ can be used.) However, there seems to be too many cases to check a priori. The following proposition will be helpful to determine whether Condition Ω_3 is satisfied or not.

Proposition 3: *Let us consider the formulation of Condition Ω_3. Let $P = \{v_1, v_2, \cdots, v_K\} = \pi(w_T)$ for some w_T. The only subsets V_i of P that need to be considered are those for which $|V_i| = K - 1$.*

Proof: Note that $V_i = P$ does not need to be considered since V_i must be a proper subset of P. For $K = 1$ the proposition is trivially true, and so is it for $K = 2$. For $K = 3$, $\{v_1, v_2\} > v_3$, $\{v_1, v_3\} > v_2$ and $\{v_2, v_3\} > v_1$ are the only links that need to be considered. Indeed, $v_1 > v_2$ does not need to be considered if $\{v_1, v_3\} > v_2$ is considered because if $\{v_1, v_3\} \not> v_2$ then $v_1 \not> v_2$, otherwise, we are done. In general, consider the following reachability cases, $V_{i_1} > v_1, \cdots, V_{i_K} > v_K$ where for any i_k, $|V_{i_k}| = K - 1$, $v_k \notin V_{i_k}$, and for any $\bar{i}_k \neq i_k$, $V_{i_k} \neq V_{\bar{i}_k}$. If for some i_k $V_{i_k} > v_k$ holds, then Condition Ω_2 is true and (v_k, w_T) can be deleted, therefore there is no need to consider the eventuality of $R > v_k$ holding for any other $R \subset P$. On the other hand, if $V_{i_k} \not> v_k$, then for any $R \subset V_{i_k}$, $R > v_k$ does not hold either. ●

Using Proposition 3, Conditions Ω_3 can be checked in a time of complexity $O(|\pi(w_T)| \times (|A_v| + |A_w|))$. The complexity of RULE 4 is therefore:

$$O\left[\left[\sum_{w_T \in W} |\pi(w_T)|\right] \times (\,|A_v| + |A_w|\,)\right]$$

$$= O\left[\,|A_v| \times (|A_v| + |A_w|)\right]$$

RULE 4:

```
for all $w_T \in W$ do
begin
   $P := \pi(w_T)$
   for all $v_i \in P$ do
   begin
      $V_i := P - \{v_i\}$
      find the set, $S$, of all the v-nodes reachable from $V_i$
      (using ALGORITHM ReachAll, section 4.3).
      if $v_i \in S$
      then begin
         REDUCE 4
         $P := V_i$
      end
   end
end
```

To show that all the redundancy featured by an F-graph can be removed by a sequential application of RULE 1-4, one must show that (i) there is no need to repeat any of RULE 1-4 more than once. (ii) RULE 1-4 are "complete", in the sense that no other rule is needed to eliminate redundancy.

Proposition 4: *Let the indexes i and j denote any two of 2, 3 or 4, and i is different from j. Given an F-graph G, an unsuccessful attempt to apply RULE i on G, may never be successful after a successful application of RULE j .*

Proof: RULE 2-4 do not alter the closure of the explicit FDs in G, and therefore do not alter reachability ($>$).●

Hence, the order in which RULEs 2-4 are applied is immaterial. For performance, since RULE 3 and RULE 4 are of the same time complexity, it makes no difference if RULE 3 precedes RULE 4 or vice versa. However, because RULE 2 is of lesser complexity than RULE 3-4, it should be applied before the latter. RULE 1 must be repeated after the application of RULE 4, and the following example shows why.

Example: Suppose the set of explicit FDs is: $\{AC \rightarrow E, ABC \rightarrow D, A \rightarrow B\}$. RULE 1 cannot be applied. However, RULE 4 can be applied and leads to: $\{AC \rightarrow E, AC \rightarrow D, A \rightarrow B\}$. Now, RULE 1 can be applied. The result is: $\{AC \rightarrow ED, A \rightarrow B\}$. •

In practice it helps to apply RULE 2 first, then RULE 1 and RULEs 3-4, then RULE 1 again, in that order. Indeed, RULE 2 is only of time complexity $O[|W|]$, and RULE 1 is of lesser time complexity than RULEs 3-4. Further, RULE 1 will eliminate most superfluous w-nodes (and the arcs incident to them or emanating from them).

The following algorithm is used to fully reduce an F-graph:

```
ALGORITHM Reduce: /* Fully reduce a user F-graph */

    RULE 2
    RULE 1
    RULE 3
    RULE 4
    RULE 1

end Reduce.
```

Proposition 5: *The application of Algorithm Reduce on an F-graph G, leads to an F-graph $\overline{G}$ which depicts a nonredundant set of explicit FDs.*

Proof: To prove the assertion of the proposition, one has to show that no explicit FD can be induced by other explicit FDs using a complete set of FD-rules. Suppose there exists in G an explicit FD, $f_r : R \rightarrow \overline{R}$, which is redundant and has not been detected by RULE 2, RULE 3 and RULE 4. Note that, from the application of RULE 2, we have $V_R \bigcap V_{\overline{R}} = \emptyset$. Because

f_r is redundant, it can be induced by a link $L(W_s, V_{\overline{R}})$ which does not contain w_r (the w-node associated with f_r) and does not contain the arcs incident to w_r or emanating from it. Since Condition Ω_2 (Proposition 2) is met, f_r should have been eliminated by RULE 3. •

Proposition 6: *Algorithm Reduce has a time complexity bound of order:*

$$O\Big[\,|A_v| \times (\,|A_v| + |A_w|\,)\Big].$$

Proof: The total time complexity of Algorithm *Reduce* is:

complexity(RULE 1) + complexity(RULE 2) + complexity(RULE 3) + complexity(RULE 4)

$$= O(|W|^2) + O(|W|) + O(|W| \times (|A_v| + |A_w|)\,) + O(|A_v| \times (|A_v| + |A_w|))$$
$$= O(\,|A_v| \times (|A_v| + |A_w|)).\bullet$$

In the next section, a correspondance is established between a reduced F-graph, i.e. an F-graph obtained as a result of the application of Algorithm *Reduce,* and a minimum cover of FDs.

4.2.2. Minimum F-graph

Algorithm *Reduce* can be used to derive an LR-minimum cover for a set of FDs as stated in the following proposition.

Proposition 7: *An F-graph is minimum if and only if it is fully reduced.*

Proof: Let us call H the set of explicit FDs in the fully reduced F-graph $\overline{G}$. Let F be the set of FDs that were explicitly represented by the original F-graph G. Algorithm *Reduce* simplifies or eliminates all the explicit FDs of F that can be derived from H using a complete set of FD rules (i.e. reflexivity, augmentation and pseudotransitivity). This implies that H is a non-redundant cover of F (Proposition 5). To show that it is minimum, we use a result reported in [MAIE 80], namely, that a non-redundant cover H is minimum if and only if there is no $X \rightarrow \overline{X}$ and $Y \rightarrow \overline{Y}$ in H such that: $X \longleftrightarrow Y$ and $X \rightarrow Y$ can be induced by using a set of explicit FDs, I, and $I \bigcap E_H(X) = \emptyset$, where $E_H(X) = \{\hat{X} \rightarrow V: \hat{X} \rightarrow V \in H \text{ and } \hat{X} \longleftrightarrow X \in H^+\}$. If for every $X \rightarrow \overline{X}$ and $Y \rightarrow \overline{Y}$ in H, $X \not\rightarrow Y$, then H is minimum. Else, if H is not minimum, suppose there exist $X \rightarrow Y$ and $Y \rightarrow X$, both in H^+, and we can reach V_Y from V_X on the F-graph of H through a set of explicit FDs I, and $I \bigcap E_H(X) = \emptyset$. This implies that there exists a set of explicit FDs $I \subset H$ which can be used to reach Y from X on the F-graph, such that for any $\hat{X} \rightarrow Z$ in I, $\hat{X} \not\rightarrow X$. If we take $\hat{X}$ to correspond to a set of v-nodes on the link from X to Y such that $\hat{X} \rightarrow Y$, then, $X \rightarrow \hat{X}$. Further, since $Y \rightarrow X$ by hypothesis, therefore $\hat{X} \rightarrow X$ too, a contradiction. Therefore, H must be minimum. Let us now show that H is also L-minimum. If it is not, then, for some explicit FD $XY \rightarrow Z$ in H, we can find an FD $X \rightarrow Z$ in H^+. This implies that there exists a link from a subset of $\sigma(V_X)$ to V_Z in $\overline{G}$ which is a contradiction since we asserted that RULE 3 and RULE 4 cannot be applied any further. Therefore H is L-minimum. Let us assume it is not LR-minimum. There must exist an FD $X \rightarrow YZ$ in H and Y is superfluous (i.e. the arcs existing between the w-node associated with the FD, and the v-nodes in V_Y, are extraneous). This implies that $X \rightarrow Y \in H^+$. Therefore, a link can be found between a subset of $\sigma(V_X)$ and V_Y, without using the w-node associated with the explicit FD $X \rightarrow YZ$. This is contradicts the fact that RULE 2 and RULE 3 cannot find a range of application. Therefore H is LR-minimum, and $\overline{G}$ is a minimum F-graph. The converse is obvious. ●

The advantages of having an algorithm such as Algorithm *Reduce* are now discussed. (1) It can be noted that Maier's algorithm [MAIE 80] is of complexity $O(n^2)$ where n is the length in number of characters of the initial set of FDs F. Since $n = |A_v| + |A_w|$, Algorithm *Reduce* seems to be of slightly lower complexity than Maier's. However, a careful look at Maier's algorithm reveals that it is really of complexity $O\left[max\left\{|A_v|\,,\,|A_w|\right\} \times n \right]$ Note that Maier's Algorithm features two steps of complexity $O(|W| \times n)$ (respectively for non-redundancy and minimality), one step of complexity $O(|A_v| \times n)$ (to enforce L-minimality), and one step of complexity $O(|A_w| \times n)$ (to enforce LR-minimality). On the other hand, Algorithm *Reduce* has only one step of time complexity $O(|A_v| \times n)$. Further, note that $|W|$ is no greater than $\frac{n}{2}$. Hence, Algorithm *Reduce* is more direct in its approach than Maier's algorithm. (2) Since the F-graph is going to be directly submitted to some processing (section 4.3-4), it is easier to use it from the beginning rather than resort to some indirect technique such as the one reported in [MAIE 80]. For instance, if the information structure of the universal database scheme is regularly updated or extended, RULES 1-4 can be applied in a straightforward manner to adapt the F-graph to a changing environment. (3) The reduction of an initial F-graph decreases considerably the complexity of the algorithms used to select appropriate CRDUs for all queries (section 4.3-4).

From now on, it may be assumed that any F-graph under consideration depicts a set of explicit FDs which is LR-minimum.

This section is closed with an example.

Example: To show the interest of considering LR-minimum covers, we depict the FDs for each FD-set below (Figure 4.4):

(1) $\{A \rightarrow B, A \rightarrow C\}$ is nonredundant, but it is not minimum since, by RULE-1 $\{A \rightarrow BC\}$ has fewer FDs.

(2) $\{ABC \rightarrow D, A \rightarrow B\}$ is minimum, but not L-minimum since, by RULE-4 the B can be removed from the left-hand-side of the first FD.

(3) $\{A \rightarrow AB\}$ is L-minimum but not LR-minimum, since by RULE-2 A can be removed from the right-hand-side.●

It has been shown above how an F-graph, depicting FDs, can be fully reduced. The next step is to enforce Assumption UR-3 if necessary.

4.2.3. Lossless F-graph

Let E be a given set of FDs. We write $E \overset{\Delta}{=} \{X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \cdots, X_n \rightarrow Y_n\}$. We now discuss how to efficiently represent the information contained in E with an F-graph. A minimum cover, F, of E can be found using Maier's algorithm [MAIE 80], and an F-graph of F can then be constructed. An equivalent approach to that of Maier is to build the F-graph of E and then to reduce it (see Section 4.2.2). Let $G = (V \bigcup W, A_v \bigcup A_w)$ be the F-graph thus obtained. We will later argue that it is desirable to depict a universal scheme $<U,F>$ with a lossless F-graph, in particular a graph of the form $L(w_0, V)$ for some $w_0 \in W$. First, we prove a Corollary which states that it is always possible to do so.

Corollary 3: *Given any F-graph $G = (V \bigcup W, A_v \bigcup A_w)$, a lossless F-graph $\overline{G}$ can be obtained from G by adding, at most, one explicit FD to G.*

Proof: By Corollary 1, if a set of relation schemes D does not possess the lossless join property with respect to a universal relation scheme $<U, >$, a new set of relation schemes $\overline{D}$ that does satisfy the lossless join property with respect to $<U, >$ can be found. Namely, $\overline{D} = D \bigcup \{<Y, >\}$, where $Y \subseteq U$ and $Y \rightarrow U$ holds. We can apply a similar argument for our case. Suppose, the F-graph G happens to be lossy. Let U denote the set of all the attributes, i.e. $U \overset{\Delta}{=} \bigcup_{i=1}^{n} X_i \bigcup Y_i$, and let V denote the set of v-nodes, i.e. $U \overset{\Delta}{=} Att(V)$. As will be shown below, we can always identify a subset S of V such that $S > V$ in G. A lossless F-graph, $\overline{G} = (\overline{V} \bigcup \overline{W}, \overline{A}_v \bigcup \overline{A}_w)$ can be obtained from G as follows: Create a new v-node $\overline{v}$ with a new attribute which is the concatenation of all the attributes of the nodes in S. Define $\overline{V} = V \bigcup \{\overline{v}\}$. Create a new w-node w_0. Define $\overline{W} = W \bigcup \{w_0\}$. Also create a new arc $(\overline{v}, w_0)$. Define $\overline{A}_v = A_v \bigcup \{(\overline{v}, w_0)\}$. Now create new arcs from w_0 to every node in S. Call this set of arcs A_o. Define $\overline{A}_w = A_w \bigcup A_o$. The new F-graph $\overline{G}$ thus obtained is of the form $\overline{G} = L(w_0, S) \bigcup G = \overline{L}(w_0, V)$, where $L(w_0, S)$ stands for $(S \bigcup \{\overline{v}, w_0\}, A_0 \bigcup \{(\overline{v}, w_0)\})$. By Corollary 2, $\overline{G}$ is a lossless F-graph. ●

If an F-graph G is of the form $G = L(w_0, V)$, for some w_0, then we say that G is *lossless in the sense of Corollary 3.* Without loss of generality, we always assume that an F-graph is lossless in the sense of Corollary 3. Indeed, if an F-graph is lossless, but not in the sense of Corollary 3, a new FD can be added as it is done for lossy graphs. Such an assumption is not essential, however it is made to simplify the exposure of the proposed design algorithms.

We now argue that no semantic constraint is either added or lost when $<U,F>$ is augmented to $<U\bigcup\{\bar{a}\},F\bigcup\{f_0\}>$. Let us denote $Att(\bar{v})$ with the new attribute name $\bar{a}$. Suppose the set S in the proof of Corollary 3 is $S = \{s_1, \cdots, s_m\}$. Let $a_1, \cdots, a_m$ denote the attribute labels of $s_1, \cdots, s_m$ respectively. We follow the following convention. $DOM(\bar{a})$ is $DOM(a_1) \times \cdots \times DOM(a_m)$. Further, the FD $f_0 : \bar{a} \rightarrow a_1 \cdots a_m$ and the identity FD $a_1 \cdots a_m \rightarrow a_1 \cdots a_m$ hold in *exactly* the same relations. Then, given some set of attributes Y such that $Y \bigcup \{a_1, \cdots, a_m\} = \bar{Y}$, the join of a relation $r \in I_{<\bar{Y},>}$ with a relation $s \in I_{<\{\bar{a}\}\bigcup Y,>}$ produces $q = r[Y]s \in I_{<\{\bar{a}\}\bigcup \bar{Y},>}$. From our convention, q contains the same tuples as $q[\bar{a}\bigcup Y)]$ or $q[\bar{Y}]$. Our convention also implies that $I_{<,F^+>} = I_{<,(F\bigcup\{f_0\})^+>}$, and thus, no semantic constraint is either added or lost when $<U,F>$ is augmented to $<U\bigcup\{\bar{a}\}\ ,\ F\bigcup\{f_0\}>$. (Equivalently, when G is augmented to $\bar{G}$). However, the database schema $\{\ <S(f),f>$ for all $f \in F\bigcup\{f_0\}\ \}$ satisfies the lossless join property, whereas $\{\ <S(f),f>$ for all $f \in F\ \}$ does not. The importance of this latter assertion will be discussed before the end of the section.

In the proof of Corollary 3 we mentioned the problem of identifying a set of v-nodes S. This problem can be modeled and solved as a simple covering problem as follows. Algorithm *NoLoss* , stated below, determines a set of w-nodes, *WsourceNodes*, that leads to all the v-nodes of the F-graph, except those in $\bigcup_{w \in WsourceNodes} \pi(w)$. The objective is to minimize the cardinality of *WsourceNodes*. By so doing, the number of attributes that belong to the key of the universal relation scheme is minimized. Algorithm *NoLoss* is formulated as a standard *covering problem.* Detailed implementation is omitted because solution techniques for this problem are well documented. Appropriate references on efficient solution techniques for the covering problem can be found in [ETCH 77, SALK 75, MURT 76].

ALGORITHM Noloss: /* Modify the F-graph, if necessary,

to make it lossless. */

```
    begin
      /* Solve the following covering problem.
        An algorithm such as [ETCH 77]'s may be used. */
    Given $p_{ij} = \begin{cases} 1 & \text{if } w_i > v_j \text{ or } v_j \in \pi(w_i) \\ 0 & \text{otherwise,} \end{cases}$

     find $x_i = \begin{cases} 1 & \text{if } w_i \in WsourceNodes \\ 0 & \text{otherwise,} \end{cases}$

    Minimize $\sum_{i=1}^{|W|} x_i$

    subject to $\sum_{i=1}^{|W|} p_{ij} x_i \geq 1$ for all $j$.

    end
    if $|WsourceNodes| > 1$ then
       begin /* Insert a new key that corresponds
                 to all the predecessors of the source nodes. */
         $W := W \cup \{w_0\}$          /* Where $w_0$
                                          is a new node. */
         $V := V \cup \{\bar{v}\}$
         $Att(\bar{v})$ is the concatenation of the attribute labels of the nodes in $\bigcup_{w \in WsourceNodes} \pi(w)$
         $A_v := A_v \cup \{(\bar{v}, w_0)\}$
         for all $v \in \bigcup_{w \in WsourceNodes} \pi(w)$ do
            $A_w := A_w \cup \{(w_0, v)\}$
       end
```

end Noloss.

In Figure 4.5-ii, G' is a lossless version of G, Figure 4.5-i.

The forthcoming theorem clarifies the dependencies between the concepts of lossless F-graph, link, and envelope. In particular, the theorem states that a lossless F-graph guarantees that, for any set of target nodes T, an envelope $K(T)$ can be determined using explicit FDs exclusively. Clearly, a lossless F-graph is desirable since one does not have to worry about inferring implicit FDs to determine query envelopes.

Theorem 2: *Consider the following five propositions where G always denotes an F-graph of the form $(V \bigcup W, A_v \bigcup A_w)$, and G depicts a universal relation scheme $<U,F>$.*

(a) *G is lossless in the sense of Corollary 3, i.e. $G = L(w_0, V)$ for some $w_0 \in W$.*

(b) *Given G, for any $T \subseteq V$, there exists a link $L(w_s, T)$ for some $w_s \in W$.*

(c) *Given G, for any $T \subseteq V$, there exists an envelope $E(T)$ such that, for any $< ,\{f_i\}> \in E(T)$, f_i is explicit in G.*

(d) *G is lossless.*

(e) *Assumption UR-3.*

The following assertions are true.

(1) (a) *if and only if* (b).

(2) (c) *if and only if* (d).

(3) *If* (a) *then* (c).

(4) *If* (e) *then* (d).

Further, if G is a minimum F-graph, then:

(5) *If* (d) *then* (e).

Proof: Assertion (1) is obvious since we are guaranteed to find at least one link with a single source i.e. $L(w_0, V)$. Assertion (2) follows immediately from the definition of a lossless F-graph and the definition of an envelope. Assertion (3) follows from (1), the definition of an envelope, and Corollary 2. All the explicit FDs of a link $L(w_s, T)$ can be picked to produce an envelope for T. Assertion (4) follows from the fact that if Assumption UR-3 holds, then, in particular, it holds for the set of relation schemes $\{<f_1>, \cdots, <f_n>\}$ where the f_i' s are all the FDs in F, and from the definition of a lossless F-graph. To prove (5), we will prove the equivalent Corollary to (5):

Corollary 4: *Suppose an F-graph G, which depicts a universal scheme $<U,F>$, is lossless and minimum. Let $\overline{G}$ be an F-graph that depicts a scheme $<U,E>$ where $E^+ = F^+$. then $\overline{G}$ is lossless.*

Proof: We know that, if for some $X_0 \subseteq U$, $X_0 \rightarrow U \in F^+$, then $X_0 \rightarrow U \in E^+$. Similarly, if for some $V_0 \subseteq V$, $V_0 > V$ is true in G, then it is also true in $\overline{G}$. Further, we claim that *such an X_0 is the left-hand-side (lhs) of some explicit FD in G.* To prove the claim we will use Corollary 1 together with the definition of a lossless F-graph. Since, by hypothesis, G is lossless, therefore, by Corollary 1, there exists an explicit FD, say f_0, such that $X_0 \subseteq S(f_0)$ and $X_0 \rightarrow U$. If f_0 is of the form f_0: $YX_0^- \rightarrow X_0^+Z$, where $X_0^- \bigcup X_0^+ = X_0$ and Y, Z are two sets of attributes, then, without loss of generality, X_0 can be redefined to be YX_0^-. This completes the proof of the claim. We now wish to show that X_0 is also in the lhs of some explicit FD in $\overline{G}$, since this, together with the knowledge that $V_0 > V$ and Corollary 1, would imply that $\overline{G}$ is minimum. We proceed as follows. Suppose X_0 is not in the lhs of any explicit FD in $\overline{G}$. Then, in $\overline{G}$, V_0 can be partitioned into at least two subsets V_0^1 and V_0^2, where $V_0 = V_0^1 \bigcup V_0^2$, and furthermore, there exist two w-nodes w_1, w_2 such that $\pi(w_1) = V_0^1$, $\pi(w_2) = V_0^2$, and finally, there exists no w-node which contains V_0 in its entirety in its predecessor set. For the rest of the proof we will use the symbol $>_G$ to denote the reachability relation in G, and $>_{\overline{G}}$ to denote the reachability relation in $\overline{G}$. In G, let w_0 be the w-node of FD f_0. Still in G, let us pick any v-node x out of the set $\sigma(w_0)$. In $\overline{G}$, suppose $w_1 >_{\overline{G}} x$. Then, $Att(V_0^1) \rightarrow Att(x) \in E^+$, therefore $Att(V_0^1) \rightarrow Att(x) \in F^+$ too. But, by hypothesis, $Att(V_0^1 \bigcup V_0^2) \rightarrow Att(x) \in F$, i.e. this latter FD is explicit in G. Therefore, by Proposition 2, G is not minimum. This latter conclusion contradicts the hypothesis. A similar argument can be applied if it is assumed that $w_2 >_{\overline{G}} x$. An identical conclusion would be obtained. Therefore both assumed assertions must be false, i.e. $w_1 \not>_{\bar{G}} x$ and $w_2 \not>_{\bar{G}} x$. Consequently, the only possibility for $V_0 > x$ to be true in $\bar{G}$ is that for some $S \subseteq \bar{W}$, where $\bar{W}$ is the set of w-nodes in $\bar{G}$, $S >_{\bar{G}} x$ and for any single $w_i \in S$, $w_i \not>_{\bar{G}} x$, and $\bigcup_{w_i \in S} \pi(w_i) \subseteq V_0$. Without loss of generality it can be assumed that $S = \{w_1, w_2\}$. Let w_y be a w-node such that $V_0^1 \not>_{\bar{G}} w_y$ and $V_0^2 \not>_{\bar{G}} w_y$, but $V_0^1 \bigcup V_0^2 >_{\bar{G}} w_y$. We claim that *there has to be at least one such w-node* w_y. If not, then for any w_y, $V_0^1 >_{\bar{G}} w_y$ or $V_0^2 >_{\bar{G}} w_y$ (we already know that $V_0^1 \bigcup V_0^2 >_{\bar{G}} w_y$ is certainly true), and in particular, for any $w_x \in \pi(x)$, $V_0^1 >_{\bar{G}} w_x$ or $V_0^2 >_{\bar{G}} w_x$, which would imply that $w_1 >_{\bar{G}} x$ or $w_2 >_{\bar{G}} x$. However, we showed earlier that this may not occur in $\bar{G}$. Thus the claim has been proved. Now let W_y be the set of all such possible nodes w_y nearest to V_0, in the sense that, for any $w_y \in W_y$, $\pi(w_y) = V_y^1 \bigcup V_y^2$, for some V_y^1, V_y^2, where $V_0^1 >_{\bar{G}} V_y^1$ and $V_0^2 >_{\bar{G}} V_y^2$. (By assumption, $V_y^1 \not>_{\bar{G}} w_y$, $V_y^2 \not>_{\bar{G}} w_y$). We claim that *the set of w-nodes* W_y *lead to the v-node x, i.e.* $W_y > x$. To prove this claim, we construct W_y as follows. Define W^0 as $\{w_1, w_2\}$, V^1 as $\sigma(w_1) \bigcup \sigma(w_2)$ and W^1 as $\bigcup_{v_i \in V^1} \sigma(v_i)$. By induction, for some integer k, we proceed as follows. Top of the construction loop: If $w_i \in W^k$ and $w_1 \not>_{\bar{G}} w_i$, and $w_2 \not>_{\bar{G}} w_i$, then change W_y to $W_y \bigcup \{w_i\}$. Continue to add new $w_i's$ to W_y whenever it is still possible to do so. If at the end of this adding process it happens that $W_y \neq \emptyset$, then terminate the construction. This is the end of the construction loop. If however, $W_y = \emptyset$, then change k to $k+1$ and let V^k be $\bigcup_{w_j \in W^{k-1}} \sigma(w_j)$, and let W^k be $\bigcup_{v_j \in V^k} \sigma(v_j)$. Go back to the top of the construction loop. It should be clear that if the construction is continued until *all* the w-nodes are examined, (i.e. not only the potential elements of W_y), then we would obtain the *largest* set of nodes w_y, say $\bar{W}_y$. Further, from the construction, $W_y >_{\bar{G}} \bar{W}_y$. Also, suppose that all the w-nodes $w_x \in \pi(x)$ are such that $w_x \notin \bar{W}_y$. Then by definition of $\bar{W}_y$, either $w_1 >_{\bar{G}} w_x$ or $w_2 >_{\bar{G}} w_x$. However, we showed earlier that this may not occur. Therefore, $w_x \in \bar{W}_y$, therefore $W_y >_{\bar{G}} w_x$ and hence $W_y >_{\bar{G}} x$. This completes the proof of the claim. Now let us redefine V_y^1 to be $\bigcup_{w_y \in W_y} V_y^1$ and V_y^2 to be $\bigcup_{w_y \in W_y} V_y^2$ (where V_y^1 and V_y^2 were defined earlier for *each* w-node w_y in W_y). From the latter claim, we have that $Att(V_y^1 \bigcup V_y^2) \rightarrow Att(x) \in E^+$ (1), and further we know that, $X_0^1 \rightarrow Att(V_y^1) \in E^+$ and $X_0^2 \rightarrow Att(V_y^2) \in E^+$ (2). Therefore, $X_0^1 \rightarrow Att(V_y^1) \in F^+$ too, and it is an implicit FD in G. Similarly, $X_0^2 \rightarrow Att(V_y^2) \in F^+$. By (1), either $V_y^1 >_G x$ or $V_y^2 >_G x$, or neither of the previous two reachability propositions are true, but yet $V_y^1 \bigcup V_y^2 >_G x$. However, suppose $V_y^1 >_G x$. Then $V_0^1 >_G x$ and, from Proposition 2, that implies that G is not minimum, since arc (w_0,x) is extraneous, which leads to a contrdiction of the hypothesis. Therefore, $V_y^1 \bigcup V_y^2 >_G x$, but $V_y^1 \not>_G x$ and $V_y^2 \not>_G x$. Since $V_0 >_G V_y^1 \bigcup V_y^2$ and $V_y^1 \bigcup V_y^2 >_G x$, therefore either the arc (w_0,x) is extraneous in G, or $V_0 = V_y^1 \bigcup V_y^2$. In any case, the hypothesis is contradicted. The latter assertion implies that X_0 is indeed on the lhs of an explicit FD in $\overline{G}$. Therefore, by Corollary 2, $\overline{G}$ is minimum. This completes the proof of the Corollary and of the Theorem. •

At this point we have developed the tools that permit us to indicate our distribution approach. For each scheduled query, an envelope will be determined. By definition, an envelope features a join dependency and therefore guarantees correctness of query processing. The FDs of all the query envelopes will be distributed; and, at each site, the FDs will be locally synthesized [BERN 76] into a database.

The first part of the distributed database design problem which consists of assigning CRDUs (mapped to FDs) to each scheduled query may now be tackled. Section 4.3 deals with the extraction of a minimum relevant sub-F-graph that suffices to answer a particular query. The material includes some techniques already reported by Lozinskii [LOZI 80]. We will discuss some pathological problems inherent in Lozinskii's algorithms, and we will propose some remedies for them. In particular, we will explain why our original definition of a link is relevant to those remedies.

4.3. Step-2: Extraction of a link

To solve a query, a relevant portion of the universal relation scheme, a query envelope, must be retained. Having shown, through Theorem 2, that it is possible to find an envelope for any set of target nodes T, and in particular, an envelope whose FDs are in some LR-minimum cover H of the user set of FDs F, we now show how to actually determine an envelope for some specific T.

Given a lossless F-graph G in the sense of Corollary 3, and given a set of target nodes T, we wish to consider the smallest set K, of relevant envelopes for T, in the following sense. For any envelope $K^i(T) \in K$, if some other envelope $K^k(T)$ is such that, $K^i(T) \subseteq K^k(T)$, then $K^k(T) \notin K$. For any T there exists at least one w-node that leads to those v-nodes. From Theorem 2, (a) is equivalent to (b), and (b) implies (c). Therefore, given a lossless F-graph, in the sense of Corollary 3, and a set of v-nodes T, to determine an envelope of T, it suffices to determine the link $L(w_s, T)$, for some w_s (indeed, any sub-link of L will do). Note that distinct w-nodes w_s will yield distinct links $L(w_s, T)$. All the envelopes contained in the set K, mentioned above, can be produced by exploiting the implicit representation of FDs in such links $L(w_s, T)$. If an envelope can be produced by using the explicit or implicit FDs of some sub-link, we will say that the envelope is *embedded* in this sub-link.

To avoid having to consider envelopes that contain extraneous FDs, some nodes w_s may have to be discarded. To that effect, Lozinskii [LOZI 80] described two short algorithms ("ALGORITHM 1" and "ALGORITHM 3-Stage B") which determine a set of w-nodes (let us call it *SourceSet*) each of which leads to a set of v-nodes T. Each w_s in *SourceSet* is characterized by the fact that $Att(\pi(w_s))$ constitutes a key for the attribute labels of T. We will refer to "ALGORITHM 1" [LOZI 80] as Algorithm *GetSource*, and to "ALGORITHM 3-Stage B" [LOZI 80] as Algorithm *GetNearestSource*. Also, Lozinskii's "ALGORITHM 2" [LOZI 80] determines the link from a given w-node w_s to a given set of v-nodes T. However, a backward pass must be added to this latter algorithm to avoid the inclusion of superfluous branches. Further, Lozinskii's algorithm does not handle potential cycling problems created by "self-reachability" (of the type $V_i > V_i$, for some $V_i \subseteq V$, other than trivially). An easy cure to this problem is to prevent any node from being a candidate, for addition in the link under construction, more than once. However, a link constructed by the algorithm would then be incompatible with Lozinskii's definition of a link, where it is possible to have $v \in \pi(w)$ and $v \in \sigma(w)$ for some v-node v and some w-node w. Our definition of a link does not allow for the representation of self-reachability, and is compatible with Lozinskii's algorithm once this algorithm is corrected as discussed. Finally, note that even if no self-reachability exists in an F-graph, Lozinskii's algorithm (augmented with a backward pass) constructs a sub-graph which may still be incompatible with Lozinskii's definition of a link. The reason is that a link, in Lozinskii's sense, is really almost the same as a proper link in our sense (it may include additional w-arcs). It is our contention that our definition is not detrimental to the generality of the proposed approach. We will refer to the corrected version of "ALGORITHM 2" [LOZI 80] as Algorithm *Getlink*. Any envelope in K must be embedded in a link $L(w_s, T)$, obtained for some node w_s in *SourceSet*.

Algorithms *GetSources*, *GetNearestSource* and *Getlink* follow. In *GetSources*, *NoPredec*(y) stands for the number of predecessor nodes of y.

ALGORITHM GetSource: (adapted from ALGORITHM 1 [LOZI 80]) /* Find all the w-nodes which lead to a set of target nodes T */

```
SourceSet := $\emptyset$
TargetNodes := $T$
for all $w \in W$ do
ReachAll
```

end GetSources.

ALGORITHM ReachAll: /* Find all the v-nodes reachable from w */

```
begin
   ReachNodes := $\pi(w)$
   NextNodes := $\{ v : (w,v) \in A_w \}$
```

```
        for all $y \in W$do
            $NoPredec[y] := |\pi(y)|$
        while $NextNodes \neq \emptyset$ and $TargetNodes \neq \emptyset$ do
        begin
            $Generate := \emptyset$
            for all $x : x \in NextNodes$
                    and $x \notin ReachNodes$ do
            begin
                for all $y : (x,y) \in A_v$ do
                begin
                    $NoPredec[y] := NoPredec[y] - 1$
                    if $NoPredec[y] = 0$ then
                      $Generate := Generate \bigcup \{v:(y,v) \in A_w\}$
                end
            end
            $TargetNodes := TargetNodes - NextNodes$
            $ReachNodes := ReachNodes \bigcup NextNodes$
            $NextNodes := Generate$
        end
        if $TargetNodes = \emptyset$ then $Source := Source \bigcup \{w\}$
    end

end ReachAll.
```

Algorithm *GetSource* is of time complexity $O(|A_v| + |A_w|)$.

ALGORITHM GetNearestSource: (adapted from ALGORITHM 3-Stage B [LOZI 80]) /*

Find the nearest subset of the set of source-nodes for a set of target nodes T */

```
    $NearestSource := \emptyset$
    for all $w \in SourceSet$ do
    begin
        $SourceSet := SourceSet - \{w\}$
        for all $\bar{w} \in SourceSet$ do
        begin
            if $w \not> \bar{w}$ then
                $NearestSource := NearestSource \bigcup \{w\}$
        end
    end

end GetNearestSource.
```

Algorithm *GetNearestSource* is of time complexity $O(|Source|^2)$.

The basis for Algorithm *GetLink* is formalized in Proposition 8.

Proposition 8: *Let $<Z,E>$ be a relation scheme, let $D = \{<X_1,\{h_1\}>, \cdots ,<X_n,\{h_n\}>\}$ be a database subschema and $H = \{h_1, \cdots ,h_n\}$ be the set of corresponding FDs defined as in the statement of Corollary 1. If D satisfies the lossless join property with respect to $<Z,E>$, then for any set of attributes $X \subseteq Z$, there exists a subschema $\bar{D} \subseteq D$, $\bar{D} = \{<X_{i_1},\{h_{i_1}\}>, \cdots ,<X_{i_k},\{h_{i_k}\}> \}$, $\bar{X} \overset{\Delta}{=} \bigcup_{j=i_1}^{i_k} X_j$, and a set of FDs $\bar{H} \subseteq H$, where $X \subseteq \bar{X} \subseteq Z$, such that $\bar{D}$ satisfies the lossless join property with respect to $<\bar{X},\bar{H}>$.*

Proof: This is a trivial consequence of Theorem 2, Assertion (2). •

In *GetLink, Forlink* is a link from w_s to v-node x, generated by a forward pass. *CurrentTarget* is a set of current target v-nodes. *CurrentNodes* is a set of nodes. To each $x \in$ *CurrentNodes* a *Forlink* which contains $\pi(x)$ has already been constructed, and x has not yet generated, i.e. no one of its outgoing arcs is contained in the current *Forlink. Successors* is a set of nodes succeeding the nodes of *CurrentNodes. Envelope* is a set of FDs that would be used to access x. *Predecessors* is a set of w-nodes preceding the nodes of *CurrentTarget. Backlink* is the initial link in a backward pass, before deletion of extra arcs and nodes. *Link*(w_s,T) stands for the final link from w_s to T. *Outlink* and *Inlink* are sets of intermediate w-nodes. "ALGORITHM 2" of [LOZI 80] was shown to be of complexity $O(|A_v| + |A_w|)$, and therefore so is the complexity of Algorithm *GetLink*.

ALGORITHM GetLink: (corrected version of ALGORITHM 2 [LOZI 80])

```
/* Forward pass to generate a link from $w_s$ to v-nodes $T$ */
Forlink := $\emptyset$
CurrentTarget := $T$
CurrentNodes := $\{w_s\}$
Successors := $\emptyset$
for all $y \in W$ do Count[$y$] := $|\pi(y)|$
for all $x \in$ CurrentNodes while CurrentTarget $\neq \emptyset$ do
begin
   for all $y : (x,y) \in A$
‡            and $y \notin W \cap$ Forlink do
```

```
        begin
            if $y \notin Successors$ then
            begin
                $Outlink := \{y\}$
                $Successors := Successors \bigcup \{y\}$
            end
            else $Outlink := \emptyset$
            if $y \in W$ then $Count[y] := Count[y] - 1$
            if $(y \in V$ or $(y \in W$ & $Count[y] = 0))$ then
            begin
                $CurrentNodes := CurrentNodes - \{x\}$
                $CurrentNodes := CurrentNodes \bigcup \{y\}$
                $Forlink := Forlink \bigcup Outlink \bigcup \{(x,y)\}$
                $Successors := Successors - \{y\}$
                $CurrentTarget := CurrentTarget - \{y\}$
            end
        end
    end
    /* End of forward pass */
    BackPass

end GetLink.
```

ALGORITHM BackPass:

```
    /* Backward pass to identify the final link from $w_s$ to $T$ */
    $ExplicitFD := \emptyset$
    $CurrentSource := \{w_s\}$
    $CurrentNodes := T$
    $Link(w_s, T) := \emptyset$
    $Backlink := Forlink$
    $Predecessors := \emptyset$
    for all $y \in W$ do $Count[y] := |\sigma(y)|$
    for all $x \in CurrentNodes$ while $CurrentSource \neq \emptyset$ do
    begin
        for all $y : (y,x) \in Backlink$ do
        begin
            if $y \notin Predecessors$ then
            begin
                $Inlink := \{y\}$
                $Predecessors := Predecessors \bigcup \{y\}$
            end
            else $Inlink := \emptyset$
            $Count[y] := Count[y] - 1$
            if $(y \in V$ or $(y \in W$ & $count[y] = 0))$ then
            begin
                $CurrentNodes := CurrentNodes - \{x\}$
                $Link(w_s, T) := Link(w_s, T) \bigcup Inlink \bigcup \{(y,x)\}$
                if $y \in W$ then $ExplicitFD := ExplicitFD \bigcup \{f(y,x)\}$
                $CurrentNodes := CurrentNodes \bigcup \{y\}$
            $Predecessors := Predecessors - \{y\}$
            $CurrentSource := CurrentSource - \{y\}$
        end
    end
end
$Link(w_s, T) := Link(w_s, T) \bigcup \pi(w_s)$
```

end BackPass.

As an example, the link $L(w_s, \{v_A, v_B, v_D, v_G\})$ obtained by the application of Algorithm *Getlink* on G (Figure 4.6. i) is shown in Figure 4.6. ii.

Note that if the original F-graph happens to be lossy, then for some given T, it may not always be possible to find a source w_s. Thus, if a set *SourceSet* can be determined for all the scheduled queries, then it is perfectly allright to use the original F-graph rather than a lossless version of it.

In the next section, a typical, but so far un-addressed, sub-problem of the distributed database design problem is identified. It is the *envelope optimization problem* (EOP) for a query, to enhance distributed query processing performance. An original formulation and solution of the problem is proposed. Problem EOP is first introduced in the form of a "brute-force" integer program (Section 4.4.2). Section 4.4.3 provides some useful ideas to solve EOP heuristically. EOP is given a more elegant model and solution in Section 4.4.4. Section 4.4.1 provides us with some useful relation-size estimation techniques from [ROSE 81, CCA 80]†.

†Also see [CHUN 83] for some possible refinements in relation-size estimation techniques.

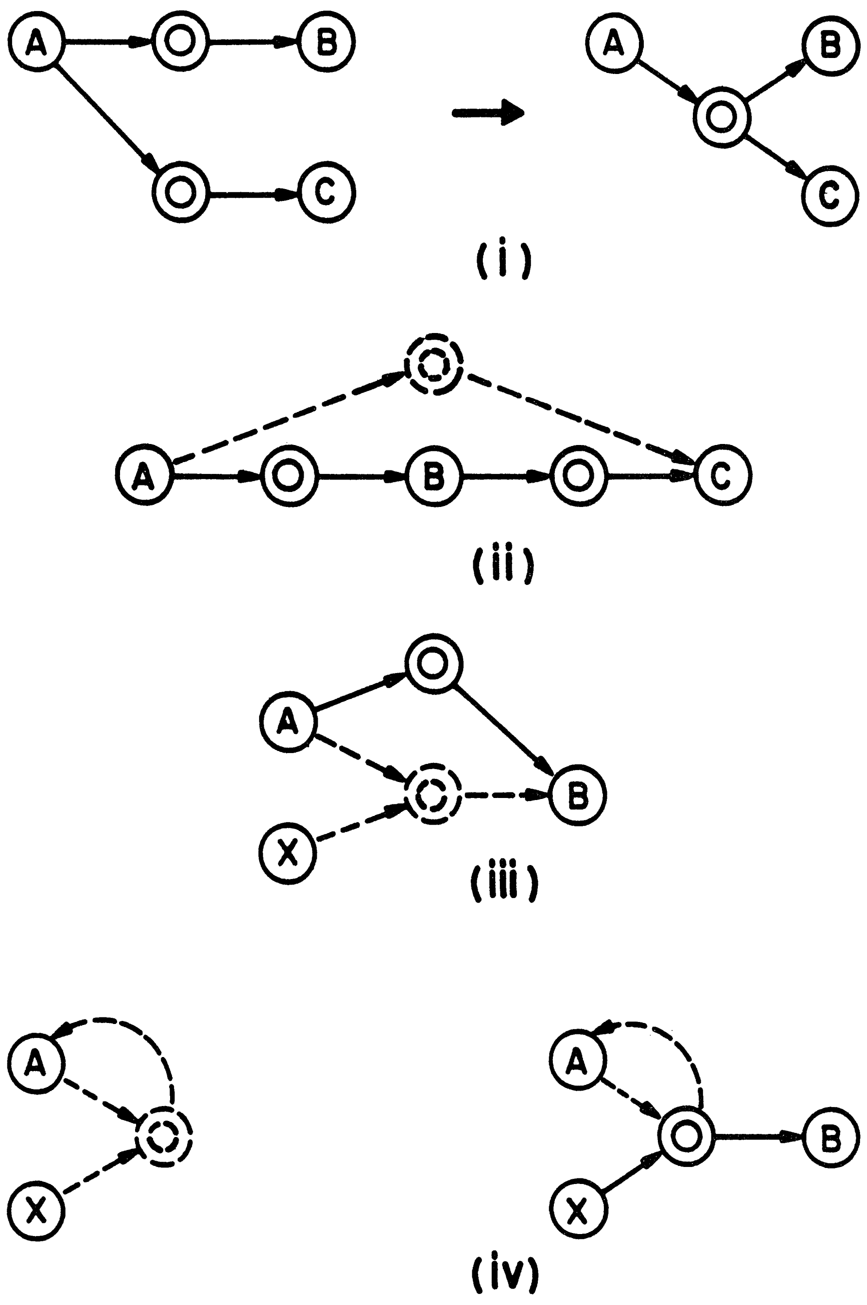


Figure 4.1. An example of F-graph reduction

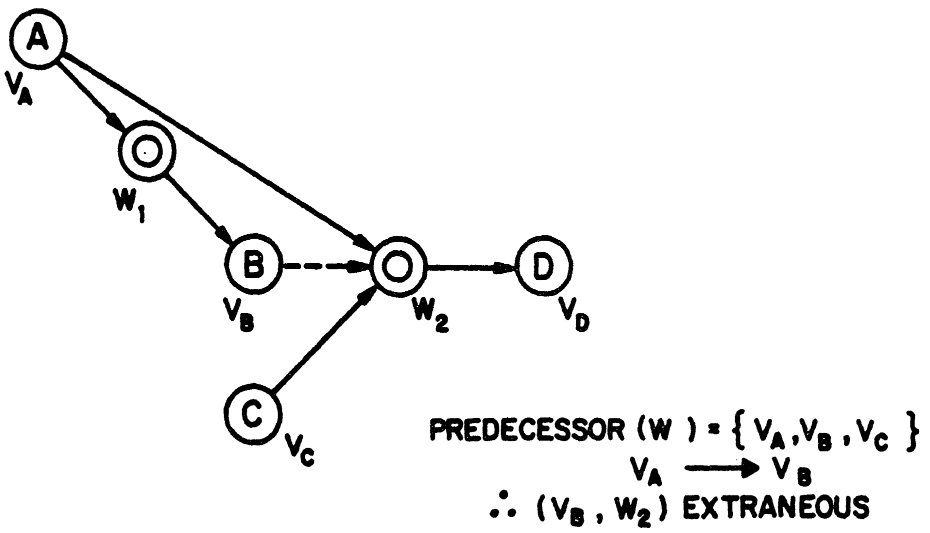

A
V_A
W_1
B
V_B
W_2
D
V_D
C
V_C
PREDECESSOR (W) = { V_A, V_B, V_C }
$V_A \longrightarrow V_B$
∴ (V_B, W_2) EXTRANEOUS

Figure 4.2. Another example of F-graph reduction

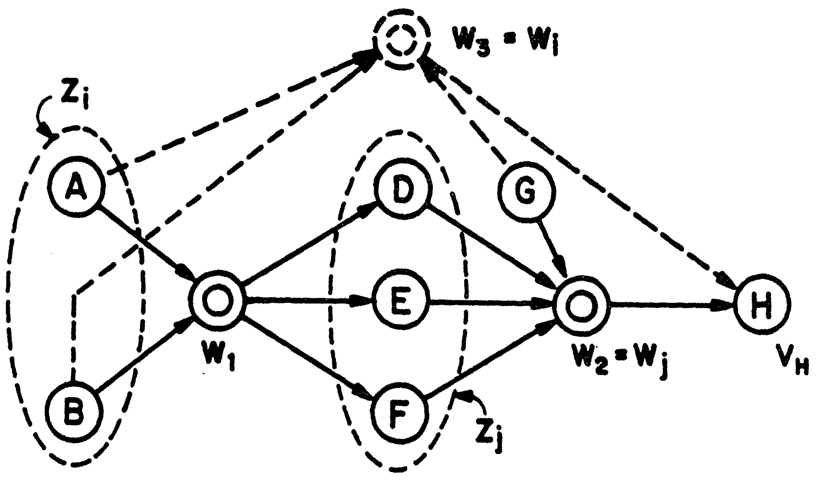


Figure 4.3. Yet another example of F-graph reduction

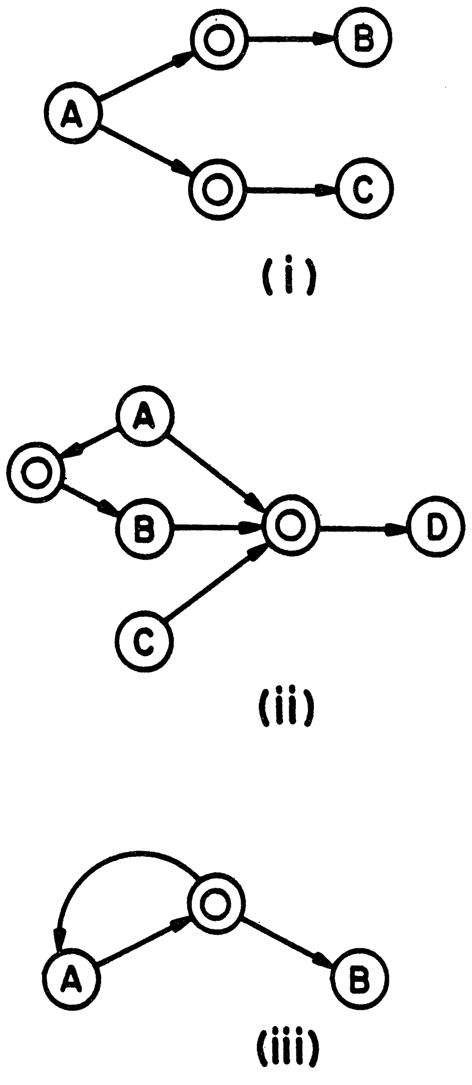


Figure 4.4. An illustration of LR-minimality

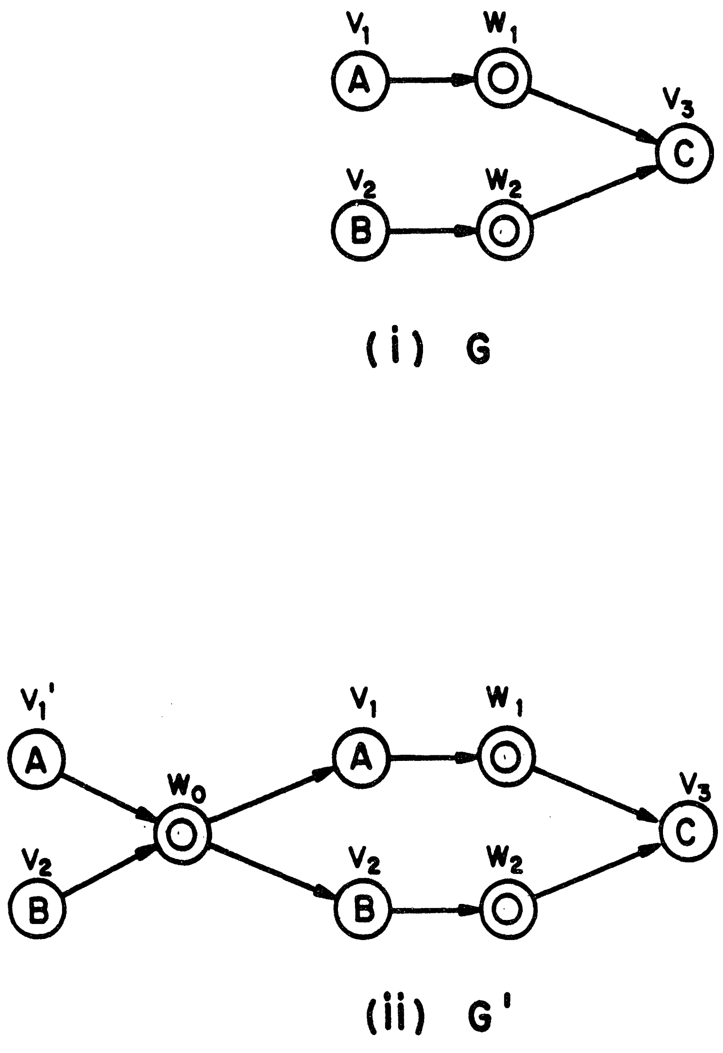
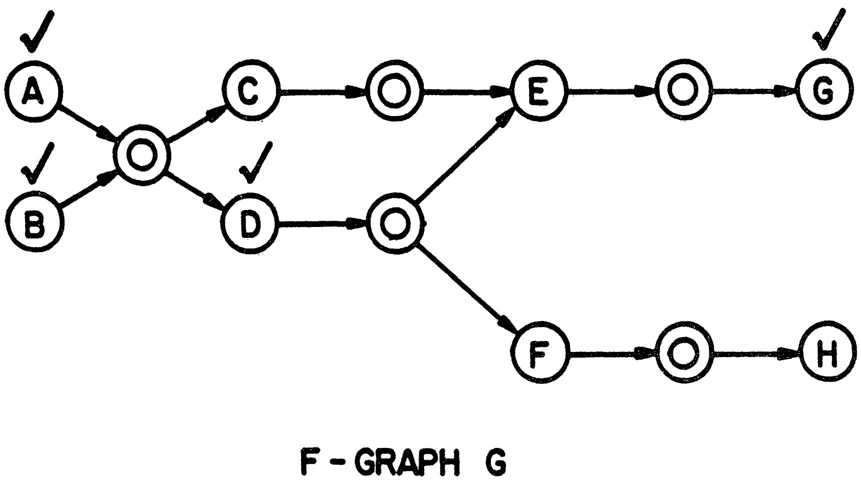
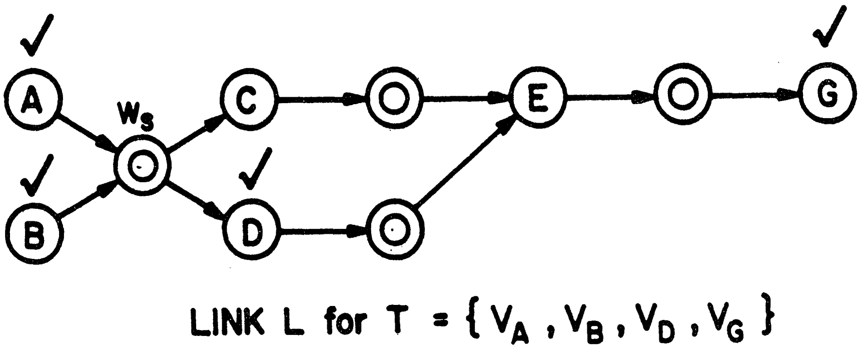


Figure 4.5. A lossy F-graph (i) and its lossless version (ii)



(i)



(ii)

Figure 4.6 A link

4.4. Step-3: The envelope optimization problem (EOP)

Given a query and its target nodes T in an F-graph, a set of sources, *SourceSet*, can be determined as discussed earlier. For each node w_s of *SourceSet*, a link $L(w_s, T)$ can also be determined (Section 4.3). The set K mentioned earlier is the set of all envelopes embedded in those links. Thus, any one of those links can safely be used to produce an envelope for the query. However, even if a node w_s in *SourceSet* is chosen arbitrarily, many options to select a valid envelope are still available. Indeed, the explicit FDs depicted in the link $L(w_s, T)$ can obviously be retained, but some other combination of implicit FDs in L could be retained as well, provided that they feature some join dependency. Thus, optimization can be used to select a "good" query envelope among all the possible ones in K. With this latter objective in mind, we state the advantages of working with a lossless F-graph in the sense of Corollary 3. (i) The explicit FDs of any sub-link that contains T can directly be used to produce an envelope of T. If the F-graph were arbitrary, we would be forced to check for the satisfaction of the lossless join property for envelopes that may contain explicit FDs as well as implicit FDs. (ii) Each of the sub-links under consideration has one single w-node for source. Since there are less sub-links of the form $L(w_s, T)$ than of the more general form $L(W_s, T)$, where W_S is a set of w-nodes, we are restricting the domain of possible envelopes in K to a reasonable size. Finally, note that in the optimization stage, we can still take advantage of the implicit representation of FDs in a sub-link, to discard extraneous attributes, by applying the transitivity inference rule (i.e. $X \rightarrow Y$ and $Y \rightarrow Z$ yield $X \rightarrow Z$).

In a distributed database environment, the following properties of a query envelope are desirable: (p-1) The number of distinct attributes is minimum. (p-2) The FDs contain as few common attributes as possible. (p-1) will tend to minimize file-to-file communication in query processing sessions. Because of (p-2), semi-joins between relations will feature high reduction capability and hence will lead to more benefit in terms of data-volume elimination. Furthermore, the overhead incurred by updates will tend to be minimized. The problem under consideration is identified as *the envelope optimization problem (EOP)*. Before tackling EOP, we will digress momentarilly to review some relevant tools for estimating the size of relations.

4.4.1. Relation-size estimation techniques

Some useful relation-size estimation techniques previously reported in [ROSE 81, CCA 80] are indicated. Although some more accurate, but more complex, techniques have been reported in [CHUN 83], we will only consider the first referenced techniques for simplicity.

The *size* of an FD, $f(w_s, v_T)$ (here, we really refer to the size of the largest relation in $I_{<S(f), f>}$), or any relation, includes the *number of tuples*, $|f|$ (the *cardinality* of f), and the *width*, $\bar{f}$. The width is simply: $\bar{f} = \sum_{v \in \pi(w_s)} \overline{Att(v)} + \overline{Att(v_T)}$, where $\bar{a}$ stands for the width (in *bytes*) of attribute a. If $f(w_s, v_T)$ is implicitly represented by a link $L(w_s, v_T)$, the number of tuples in $f(w_s, v_T)$ is the number of tuples in the relation obtained by joining all the FDs explicitly represented in L, and by projecting over $Y = \left[\bigcup_{v \in \pi(w_s)} \{Att(v)\} \right] \bigcup \{Att(v_T)\}$.

To estimate this number of tuples, the same assumption as in [ROSE 81]'s is made, namely,

Assumption Join [ROSE 81] **:** *For each value $x \in DOM(X)$ of some attribute X, the number of tuples of $<Y,>$ such that $t[X] = x$ and $X \subset Y$, determines a random variable $n_x(X)$. For some other relation scheme $<W,>$ such that $Z \subset W$, for some Z, and $DOM(X) = DOM(Z)$, $n_x(Z)$ is defined similarly. $n_x(X)$ and $n_x(Z)$ are independent. Furthermore, all the relation schemes $<Y, f[Y]>$ explicitly represented in a link L are such that* $E[n_x(X)] = \frac{|f[Y]|}{|DOM(X)|}$.

From [ROSE 81] and Assumption Join, given two relations $R[X]$ and $S[Y]$, such that $X, Y \subseteq U$ and $X \bigcap Y = Z$, the cardinality of their join $R[Z]S$ is given by $|R[Z]S| = |R| \times |S| / |DOM(Z)|$. From [CCA 80], given a relation $R[X]$, the cardinality of its projection over $Z \subseteq X$, $R[Z]$, is given by: **If** $Z = \{A\}$, $|R[Z]| = |R[A]|$, *for some* $A \in U$,

else if $\prod_{A \in Z} |R[A]| < |R|$, **then** $|R[Z]| = \prod_{A \in Z} |R[A]|$, **else** $|R[Z]| = |R|$.

The first formulation of EOP follows.

4.4.2. A brute-force approach to solve EOP

The purpose of this section is not to solve EOP, but rather, to get a feel for possible heuristic solutions for it. To tackle the *envelope optimization problem,* it is assumed that the following items are given:

SourceSet	A set of source w-nodes for a set of target v-nodes T.
$L(w_s, T)$	Link from w-node w_s to T for every $w_s \in$ *SourceSet*. (In Sections 4.4.2 and 4.4.3, and only those sections, it will be assumed that a single node w_s is picked from the set *SourceSet*, and that the link $L(w_s, T)$ rooted in w_s is selected. How to determine such a w_s will be discussed in Section 4.4.4.1).
k_{hi}	$= \begin{cases} 1 \text{ if } v_h \text{ is adjacent to } w_i \text{ in } L, \\ 0 \text{ otherwise.} \end{cases}$
$\|f\|$	Number of tuples in any base-FD f of the F-graph.
$\bar{f}$	Number of *bytes* per tuple of any base-FD f of the F-graph.
$\|f[a]\|$	Number of distinct values of attribute a belonging to the attribute-scheme of any base-FD f of the F-graph.
$\bar{a}$	Width in bytes of attribute type a.
$\|DOM(a)\|$	Number of distinct values of attribute type a.

The following notation is used:

f_{ij}	A base-FD in H^+ which is implicitly represented in L, by w_i and v_j, such that: $f_{ij} = \bigcup_{v \in \pi(w_i)} \{Att(v)\} \rightarrow Att(v_j)$.

The following 0-1 variables are to be determined:

u_{ij}	$= \begin{cases} 1 \text{ if } w_i \in W_L \text{ is associated with } f_{ij} \\ 0 \text{ otherwise.} \end{cases}$ *
U_{ijkl}	$= \begin{cases} 1 \text{ if } f_{ij} <\| f_{kl} \text{ is a semi-join to be included,} \\ 0 \text{ otherwise.} \end{cases}$

*Algorithm *GetAllFDs*, to follow, determines if u_{ij} can be equal to 1.

The problem of envelope optimization for a query is initially addressed as follows:

EOP: Given the above information, determine an envelope $K(T)$ so that:

(A) The number of attributes contained in $K(T)$ is a minimum.

(B) No FD in $K(T)$ is extraneous. In particular, no FD in $K(T)$ can be inferred from other FDs in $K(T)$ by transitivity†.

(C) An objective function which is a function of the data-volume flow is minimized. Note that possible advantages associated with the sequencing of semi-joins for query processing are not considered. This is to keep the overhead of the solution techniques for EOP to a minimum.

In this section, the following assumptions are made to evaluate the cost of item (C):

Assumption EOP-cost-1: *Query processing startegy: Each FD is assigned to a CRDU. The CRDUs are first locally processed to eliminate unwanted tuples. All pairwise semi-joins are evaluated.*

Assumption EOP-cost-2: *Since the flow of data-volume corresponding to the answer is generally small compared to the file-to-file data-volume flow, only the latter is considered.*

The symbol Z_{ijkl} is defined to be the "worst-case" overhead cost of semi-joins $f_{ij} < X] f_{kl}$, and $f_{kl} < X] f_{ij}$, where $X \overset{\Delta}{=} S(f_{ij}) \bigcap S(f_{kl})$ as:

$$Z_{ijkl} \overset{\Delta}{=} \min \left\{ Z^{\rightarrow}{}_{ijkl} \ , \ Z^{\leftarrow}{}_{ijkl} \right\}$$

where,

$$Z^{\rightarrow}{}_{ijkl} = |f_{ij}[X]| \times \bar{X} \ ^{\ddagger}$$

†The idea is to avoid unnecessary increase in semi-join occurence and therefore of cost. See properties (p-1) and (p-2) for (A) and (B).

$$+ \bar{f}_{kl} \times |f_{kl}| \times \frac{|f_{ij}[X]|}{|DOM(X)|} \ ^{\dagger}$$

$$+ \overline{f_{kl}[X]} \times |f_{kl}[X]| \times \frac{|f_{ij}[X]|}{|DOM(X)|} \ ^{\ddagger}$$

$$+ \bar{f}_{ij} \times |f_{ij}| \times \frac{|f_{ij}[X]| \times |f_{kl}[X]|}{|DOM(X)|^2} \ . \ ^{\dagger}$$

The terms with a "†" superscript stand for the assembly flow costs, and the terms with a "‡" superscript stand for the reducing flow costs.

By Assumption EOP-cost-2,

$$= |f_{ij}[X]| \times \left[\bar{X} + \frac{\overline{f_{kl}[X]} \times |f_{kl}[X]|}{|DOM(X)|} \right].$$

The overhead cost of semi-join $f_{kl} < X] f_{ij}$, or $Z^{\leftarrow}{}_{ijkl}$, is defined similarly.

An integer program for the envelope optimization problem is stated as follows:

$$EOP\text{: } Minimize \sum_{\substack{w_i, w_k \in W_L \\ v_j, v_l \in V_L}} Z_{ijkl} \ U_{ijkl} \qquad \text{(eop-0)}$$

Subject to:

Assure the covering of the target nodes.

$$\sum_{i=1}^{|W_L|} u_{ij} \geq 1 \qquad \text{(eop-1)}$$

$$\forall \ j : v_j \in T.$$

The attributes of the source are covered.

$$\sum_{j=1}^{|V_L|} u_{ij} \geq 1 \qquad \text{(eop-2)}$$

$$\forall \ i : w_i \in SourceSet$$

If a semi-join is retained, then the FDs involved in it must be included in the envelope.

$$k_{jk}\, u_{ij} + k_{jk}\, u_{kl} - U_{ijkl} \leq 1 \qquad \text{(eop-3)}$$
$$\forall \quad i, j, k, l.$$

Transitivity-redundancy is undesirable.

$$u_{ij}\, k_{j\bar{j}}\, u_{\bar{j}k}\, u_{ik} = 0 \qquad \text{(eop-4)}$$
$$\forall \quad i, j, \bar{j}, k.$$

Connectedness: For all v-nodes v_j such that $u_{ij} = 1$ for some w-node w_i, there exists a chain† (Algorithm *CheckConnect*) from *SourceSet* to v_j.

Connectedness. (eop-5)

This integer program can be solved with a branch-and-bound algorithm. General guidelines to design such an algorithm are discussed below. Specific details of implementation are however omitted because although an optimal optimization technique can be applied and is manageable in the context of solving EOP, it is obvious that it would incur a large overhead in processing time because all queries must be considered individually. Therefore a heuristic approach to solve EOP will be proposed later. The branch-and-bound solution would proceed as follows.

EOP-ip-1: Solve EOP without constraints (eop-4) and (eop-5). This is an assignment problem, therefore the Hungarian method [MURT 81], may be used to compute lower bounds. If (eop-4) and (eop-5) happen to be satisfied in the solution, then the problem has *fathomed.* If the problem has fathomed and this step is being visited for the first time, then we are done. Else, if the cost of the fathomed problem is less than the cost of the *incumbent,* where *incumbent* is initialized to be $\emptyset$ and its cost to be ∞, then the current problem takes the place of the incumbent. Otherwise it is *pruned.* We then go to EOP-ip-2.

†A directed path.

EOP-ip-2: Branch on a variable u_{ij} (set $u_{ij} = 0$, *or* $u_{ij} = 1$) to generate two new candidate problems. A LIFO search may be used. *Pruning* occurs when a candidate problem is infeasible, or when its lower bound is larger than the incumbent's. If there are still some *dangling* candidate problems, then go back to EOP-ip-1. Else, go to EOP-ip-3.

EOP-ip-3: The optimal solution is the incumbent.

Given a link $L(w_s, T)$, to find out which are the acceptable variables u_{ij}, the following algorithm may be used.

ALGORITHM GetAllFDs: /* Given a link $L(w_s, T)$ of G, find all the FDs, with a single attribute on the right-hand-side, which are explicitly or implicitly represented. */

```
$Wstart := SourceSet$
$FDset := \emptyset$
$Vinter := \emptyset$
for all $w_i \in Wstart$ while $Wstart \neq \emptyset$ do
begin
    /* Find the v-nodes $V_i$ reachable from $Wstart$ */
    $TargetNodes := V$
    $V_i := ReachAll(w_i)$
    for all new $v_j \in V_i$
        $FDset := FDset \bigcup \{u_{ij}\}$
    $Wstart := Wstart - \{w_i\}$
    $Vinter := \sigma(w_i)$
    for all $v_k \in Vinter$ do
        $Wstart := Wstart \bigcup \sigma(v_k)$
end
```

end GetAllFDs.

Algorithm *GetAllFDs* is of time complexity bound $O(|A_{v_L}| + |A_{w_L}|)$.

To check the connectedness condition (constraint (eop-5)), the following algorithm can be used:

ALGORITHM CheckConnect: /* Given a set of FDs explicitly or implicitly represented in a link L, determine if they satisfy the lossless join property. */

```
Valid := true
Vcurrent := $\emptyset$
Wcurrent := $\emptyset$
Wstart := $\{w_i \in W_L : u_{ij} = 1$ for some $v_j \in V_L\}$ - Wsource
for all $w_i \in$ Wstart while Valid = true do
begin
   for all $v_j \in \pi(w_i)$ and $v_j \notin$ Vcurrent
   while Valid = true do
   begin
      Vcurrent := Vcurrent $\bigcup \{v_j\}$
      if $\forall$ $k$, $u_{kj} = 0$ then Valid := false
   end
end

end CheckConnect.
```

Algorithm *CheckConnect* is of time complexity $O(|W|^2)$.

In the next section a heuristic approach to solve the envelope optimization prolem EOP is suggested and discussed. The primary purpose of Section 4.4.3 is to motivate the final formulation and solution of EOP in Section 4.4.4.

4.4.3. A first heuristic approach to solve EOP

There are two immediate reasons for considering a heuristic technique rather than mathematical optimization for solving EOP. (1) The exact closed-form objective function is almost impossible to obtain because the partitioning of the query envelope is to be known in advance before the distributed query processing strategies can be determined. (2) It is not worth using a mathematical programming technique such as branch-and-bound because it is too expensive. Further, looking for a "true" optimal solution is arguable after making Assumptions eop-cost-1 and eop-cost-2. The heuristic approach is now discussed.

Given is a link $L(w_s, T)$ where w_s is a source of T the set of target nodes. There is no redundancy** and no extraneous information†. Furthermore, the link is the shortest possi-

**Because the link was extracted from a minimum F-graph.

†By definition of a link.

ble‡. The goal, in Step-3, is to get a minimum cost lossless FD-covering†† of X.

One possible approach is proposed below and discussed.

APPROACH 3-a: A first heuristic for (EOP).

Step-3-a-1: Given the link $L(w_s, T)$, define $W_T \overset{\Delta}{=} \bigcup_{v_T \in T} [\sigma(v_T) \bigcap W_L]$. For each pair of nodes w_i, w_j in $\mathbf{Y} = W_T \bigcup \{w_s\}$ ‡‡ find all the proper links from w_i to $\pi(w_j)$.

Step-3-a-2: Augment the proper links to create proper links which explicitly represent the FDs which were previously implicitly represented (by exploiting transitivity).

Step-3-a-3: Construct a graph $(\mathbf{Y},\mathbf{L})$ such that any edge in $\mathbf{L}$ is associated with a proper link. Find a minimum spanning out-tree (also referred to as *arborescence*) of graph $(\mathbf{Y},\mathbf{L})$.

The steps of Approach 3-a are discussed below.

4.4.3.1. Step-3-a-1: Proper link distinguishing

Algorithm *Distinguish*, which follows, finds all the proper links between a w-node w_s and a v-node v_T. Algorithm *Distinguish* branches backwards through the link from v_T (*Backtrack*). It pushes in *STACK* the branches traced from a current set of w-nodes, *VISIT*. Branching is interrupted whenever a v-node v_m is found to feature more than one w-node predecessors. When this condition is detected, a "\$" character is pushed in the stack, v_m is labelled *DANGLE*, and the link is branched through another live branch starting from any w-node in $LiveW[v_m]$ (where $LiveW[v_m]$ is the set of predecessors w-nodes of v_m which can still be back-tracked from, at this stage in the algorithm). When the source of the link, w_s, is

‡By virtue of Algorithm *GetNearestSource*.

††With FDs which are either *explicitly* or *implicitly* represented in the link under consideration.

‡‡We assume for the time being that each node v_T of T has at most one successor w_T in L. This assumption will be relaxed in section 4.4.4.

reached by all the branches generated in *Backtrack*, one new proper link $\bar{L}^k$ is found. The stack is popped until a **$** is retrieved, and if some v-nodes are still labelled *DANGLE*, a new branch can be traced back to create a new proper link. The algorithm proceeds in this manner until all the possible combinations are exhausted. Algorithm *Distinguish* follows:

ALGORITHM Distinguish:

```
    $k := 1$ /* Number of proper links */
    $LiveW := \emptyset$
    $VISIT := \pi(v_T)$
    $\bar{L}^k := \{v_T\}$
    $PUSH(v_T)$
    if $|\pi(v_T)| > 1$ then
    begin
        $DANGLE := \{v_T\}$
        $w_0 := \pi(v_T)$
        $Succ(w_0) := v_T$
    end
    else do begin
        $DANGLE := \emptyset$
        $PUSH(\ \$\ )$
        $LiveW[v_T] := \pi(v_T)$
        $w_0 := w_i \in LiveW[v_T]$
        $Succ(w_0) := v_T$
    end
    $Backtrack(w_0)$
    while $DANGLE \neq \emptyset$ do
    begin
        $Retract(w_j)$ /* Where $w_j$ is outputted by Retract */
        $Backtrack(w_j)$
    end
```

end Distinguish.

ALGORITHM Backtrack(w_i):

```
    if $k > 1$ then $k := k + 1$
    for all $w_i \neq w_s \in VISIT$ while $VISIT \neq \{w_s\}$ do
    begin /* Backtrack terminates when $w_s$ is found and there
          is no more dangling branch */
        $PUSH(w_\perp)$
        $\bar{L}^k := \bar{L}^k \bigcup \{w_i, <w_i, Succ(w_i)>\}$
        for all $v_m \in \pi(w_i)$ do
        begin
            $PUSH(<v_m, w_i>)$
            $PUSH(v_m)$
        $\bar{L}^k := \bar{L}^k \bigcup \{<v_m, w_i>, v_m\}$
        if $|\pi(v_m)| > 1$ then
        begin
           $PUSH(\$)$
           $LiveW[v_m] := \pi(v_m)$
           $VISIT := VISIT \bigcup (w_j : w_j \in \pi(v_m))$
           $Succ(w_j) := v_m$
           $DANGLE := DANGLE \bigcup \{v_m\}$
        end
        else do begin
           $VISIT := VISIT \bigcup \{w_j = \pi(v_m)\}$
           $Succ(w_j) := v_m$
        end
     end
     $VISIT := VISIT - \{w_i\}$
  end
  $\bar{L}^k := \bar{L}^k \bigcup \pi(w_s) \bigcup_{v \in Predecessor(w_s)} (v, w_s)$


end Backtrack.
```

ALGORITHM Retract(w_j):

```
  $Item := \dagger$  /* Dummy character */
  while $Item \neq \$$ and $STACK \neq \emptyset$ do
  begin /* Have to find at least one $ */
     $POP(Item)$ /* Retrieve a character from the stack in Item */
     $\bar{L}^{k+1} := \bar{L}^k - \{Item\}$
     if $Item = \$$ then
     begin
        $v_m := ReadTop(STACK)$
        if $LiveW[v_m] = \emptyset$ then
        begin
           $DANGLE := DANGLE - \{v_m\}$
           $Item := \dagger$
        end
        else begin
           $VISIT := VISIT \bigcup \{w_j : w_j \in LiveW[v_m]\}$
           $LiveW[v_m] := LiveW[v_m] - \{w_j\}$
           $PUSH(\$)$
           $\bar{L}^{k+1} := \bar{L}^{k+1} \bigcup \{<w_j, v_m>, w_j\}$
        end
     end
  end


end Retract.
```

We now show that *Distinguish* accomplishes what it is supposed to.

Proposition 9: *A sub-link $\bar{L}(w_s, v_T)$ is proper, if and only if any v-node in it has at most one predecessor w-node.*

Proof: Let $\bar{L}(w_s, v_T)$ be a proper link. We proceed by contradiction. Assume that there exists $z \in V_{\bar{L}}$ and $X, Y \subseteq V_{\bar{L}}$ and two w-nodes w_X and w_Y, such that $\pi(w_X) = X$ and $\pi(w_Y) = Y$. Then there exist two sub-links of length one, $\bar{L}_X(w_X, z)$ and $\bar{L}_Y(w_Y, z)$ and thus, any one of the arcs (w_X, z) or (w_Y, z) is extraneous. The latter conclusion contradicts the assumption that $\bar{L}$ is proper. Conversely, given a sub-link $L(w_s, v_T)$, if for any v-node z there exists at most one predecessor w-node, then by branching backwards from v_T in L, while preserving reachability, all the nodes and arcs must be retained to depict the relationship $\pi(w_s) > v_T$. Therefore L satisfies the definition of a proper link. •

It is obvious that Algorithm *Distinguish* finds all the sub-graphs of $L(w_s, v_T)$ which are "rooted" in w_s and which include v_T. Since they all satisfy the condition of Proposition 9, therefore they are proper as well. Since the algorithm enumerates all the possibilities by exploiting the condition of Proposition 9, therefore it keeps track of all the proper links between w_s and v_T. The number of proper links generated by Algorithm *Distinguish* is larger than $\bar{P}^d$ where $\bar{P}$ is the average number of w-nodes in $\pi(v_i)$ over all v-nodes v_i, and d is the length of the link under consideration. It is a *non polynomial* algorithm, but can be used as long as $\bar{P}^d$ is not too large.

4.4.3.2. Step-3-a-2: Proper link augmenting

For each FD $f(w,v)$ implicitly represented by a pair (w,v) in a proper link $\bar{L}(w_s, v_T)$, a new FD can be explicitly represented by adding a new arc (w,v). Since $\bar{L}$ is proper, therefore $\bar{L}(w_s, v_T) \bigcup \{(w,v)\} - L(w,v)$ is also proper. In Step-3-a-2 the idea is to create new explicit FDs (exploiting transitivity on $\bar{L}$) while deleting intermediate links. Note that since each v_T

of T is considered individually in this manner, whether another target node, say $\bar{v}_T$, belongs to the link $L(w,v)$ being dropped or not, is irrelevant.

4.4.3.3. Step-3-a-3: Spanning out-tree

In this step, an algorithm to find a spanning out-tree rooted at one node may be used. Such an algorithm is described in [LAWL 76]. In general, the algorithm must be able to detect the existence of cycles in a graph, which makes it inefficient. However, in this context, it will be shown in Section 4.4.4.2 that drastic simplifications can be made.

4.4.3.4. Remarks on the initial heuristic

Several remarks can be made about Approach 3-a. (1) The problem of finding all the proper sub-links between a pair of nodes is np-complete because it can be restricted to the well known *path distinguisher problem* [GARE 79]. However, this problem may still be manageable for many practical queries because redundancy has been eliminated in Step-1, and because the link under consideration has been shortened to a maximum in Step-2. (2) Any query envelope must be determined on an underlying sub-graph of (**Y**,**L**), such as an out-tree, which is rooted in w_s. This comes from the fact that any envelope must satisfy the lossless join property (see also constraint (eop-2)). (3) The sub-graph must at least be a tree, i.e. it must be connected, because there must exist a link from the root node, w_s, to any other node, even if this link is not represented by a single edge in (**Y**,**L**). This also follows from the lossless join property (see also constraint (eop-5)). (4) The sub-graph may be more than an out-tree, but this would add unnecessary "access paths", and thus increase data redundancy. This remark agrees with requirements (A) and (B) (see also constraint (eop-4). Note, however, that constraint (eop-4) is extraneous in problem EOP for it tends to decrease the objective function). (5) Requirement (A) is obviously satisfied, further, all the target nodes are covered (see constraints (eop-1) and (eop-3)). (6) A spanning tree may be more than needed since the initial link may or may not be proper. Eventually, some branches of the spanning tree may be thrown out. We will formalize those remarks in the next section (Proposition 16).

Since the first heuristic avoids the penalty of using branch-and-bound by introducing some potential curse of cardinality implied in Step-3-a-1, as well as some substantial overhead in Step-3-a-2, a more direct solution approach to problem (EOP) is obviously called for. It is presented in the next section.

4.4.4. Final solution of EOP

An immediate simplification to the first heuristic is to restrict the types of proper links to be determined in Step-3-a-1 and 3-a-2. This suggests the following heuristic approach.

APPROACH 3-b: A refined heuristic for (EOP)

Step-3-b-1:

Select w_s in *SourceSet* such that it yields the shortest proper link $\bar{L}*(w_s, T) \overset{\Delta}{=} (V_{\bar{L}*} \bigcup W_{\bar{L}*}, A_{V_{\bar{L}*}} \bigcup A_{W_{\bar{L}*}})$ to T. Construct the *Metagraph* (**Y**,**L**) of $\bar{L}*$ as outlined through the following algorithm.

ALGORITHM Metagraph:

```
/* Note that the only nodes and arcs considered are those
  of $\bar{L}*$, and that $\pi$ and $\sigma$ are restricted to $\bar{L}*$. */
L := $\emptyset$ /* Arcs */
Y := $\{<\pi(w_s), w_s>\}$ /* Nodes */
for all $v_T \in T$ do
begin
   if $\sigma(v_T) \neq \emptyset$ then
   begin
      for all $w \in \sigma(v_T)$ do
      Y := Y $\bigcup \{<\pi(w), w>\}$
   end
   else Y := Y $\bigcup \{<\{v_T\}, \text{nil}>\}$
end
for all $<V_i, w_i>$ and $<V_j, w_j> \in$ Y do
```

```
begin
   if $\bar{L}*(w_i, V_j)$ exists, then
   L := L $\bigcup$ {[< $V_i$, $w_i$ > , < $V_j$, $w_j$ >]}
   if $\bar{L}*(w_j, V_i)$ exists, then
   L := L $\bigcup$ {[< $V_j$, $w_j$ > , < $V_i$, $w_i$ >]}
end
if there exists $<\bar{V},w> \in$ Y and $<V,\bar{w}> \in$ Y : $\bar{V} \subseteq V$
then add the arc [$< V,\bar{w}>$ , $<v,w>$] to L.

end metagraph.
```

Step-3-b-2:

For each arc $e = [<V,w> , <\bar{V},\bar{w}>] \in$ **L**, estimate the size of the FD $f(w,\bar{V})$ induced by $\bar{L}*(w,\bar{V})$. Define the weight of e to be the size of $f(w,\bar{V})$. In particular, if $\bar{V} \subseteq V$, then the weight of e is zero.

Step-3-b-3:

Find a minimum spanning out-tree of graph (**Y**,**L**).

Each step of Approach 3-b will now be discussed. Note first that remarks (2) through (5) of Section 4.4.3.4 still hold for Approach 3-b with the new definition of (**Y**,**L**). However, remark (6) can be changed as follows. (6) A spanning tree *is* needed because the initial link is a proper link, and therefore no node can be thrown out of **Y**. (This is consistent with property (p-2) of an envelope, Section 4.4).

4.4.4.1. Step-3-b-1: Minimum proper link

Algorithm *MinPropLink* which follows, finds all the minimum length proper link between w_s and $v_T \in T$. It is very similar to the classical *shortest path* algorithm of Dijkstra [MURT 76]. The algorithm labels each v-node of the initial link as *UNLAB, PERM* or *TEMP**. At the beginning, *TEMP* is empty, while *UNLAB* contains all the intermediate v-nodes of the link (V_L - $\pi(w_s)$), and *PERM* contains $\pi(w_s)$. The link is branched out from w_s in search of a minimum length proper link that leads to v_T. Nodes of *UNLAB* are branched

*They stand for *unlabelled, permanently labelled* and *temporarily labelled.*

from one by one, and transferred into *TEMP*. A v-node v_j in *TEMP* is always a candidate for branching until v_T is reached. When a minimum branch from *PERM* to *TEMP* is selected, the v-node y which is at the end of the branch is transferred into *PERM* and is never visited again. When v_T is found, the algorithm traces back the graph from v_T using the *Predec* labels. The same process is repeated for every v_T in T. Algorithm *MinPropLink* follows:

ALGORITHM MinPropLink:

```
for all $v_T \in T$ do /* $T$ is the set of target nodes */
    $UNLAB := V_L - \pi(w_s)$
    $PERM := \pi(w_s)$
    $TEMP := \emptyset$
    for all $v_i \in PERM$ do
    begin
        $Predec[v_i]$ := nil /* Initialize the predecessor pointers */
        $Length[v_i] := 0$ /* Length of the shortest directed path from PERM */
    end
    $Length[w_s] := 0$
    for all $v_j \in \sigma(w_s)$ do
    begin
        $TEMP := TEMP \bigcup \{v_j\}$
        $UNLAB := UNLAB - \{v_j\}$
        $Predec[v_j] := w_s$
        $Length[v_j] := 1$
    end
/* At this stage, PERM and TEMP have been initialized */
    while $v_T \notin PERM$ do /* Quit whenever $v_T$ is reached */
    begin
        if $TEMP \neq \emptyset$ then
        $y := v_j : Length[v_j] = \min \{Length[v_k] : v_k \in TEMP\}$
    †   $PERM := PERM \bigcup \{y\}$ /* Add in PERM the node $y$ from TEMP which
        is nearest to PERM than any other node in TEMP. */
        for all $w_j \in \sigma(y)$ do
            $Length[w_j] := \max\{Length[v_k] : v_k \in \pi(w_j)\}$
        $TEMP := TEMP - \{y\}$
        for all $v_j \in TEMP$ do /* Update the labels of all the nodes in TEMP */
        begin
            if ( there exists $w_j : \pi(w_j) \subseteq PERM$
                and $v_j \in \sigma(w_j)$,
                and $w_j \neq Predec(v_j)$,
                and $Length[v_j] > Length[w_j] + 1$ )
            then begin
                $Length[v_j] := Length[w_j] + 1$
                $Predec[v_j] := w_j$
                $Length[w_j] := \max \{Length[v_k] : v_k \in \pi(w_j)\}$
            end
        end
        for all $v_j \in UNLAB$ do /* Update $TEMP$ */
        begin
            $Predec[v_j] := w_j$ such that:
            $Length[w_j] = \min \{Length(w_k)$ where $\pi(w_k) \subseteq PERM$ & $v_j \in \sigma(w_k)\}$
            $Length[v_j] := Length[w_j]$
            $UNLAB := UNLAB - \{v_j\}$
            $TEMP := TEMP \bigcup \{v_j\}$
        end
    end /* of the while loop */
  /* The minimum length is found by tracing back the graph from $v_T$
    using the $Predec$ labels. */
    $\bar{L}_T * := \{v_T \llcorner Predec(v_T), <Predec(v_T), v_T>\}$
    for all $w_j \in \bar{L}_T *$ while not a link do
    begin
        $\bar{L}_T * := \bar{L}_T * \bigcup \pi(w_j)$
        $\bar{L}_T * := \bar{L}_T * \bigcup_{v \in \pi(w_j)} <v, w_j>$
        for all $v_j \in \pi(w_j)$ do
            $\bar{L}_T * := \bar{L}_T * \bigcup \pi(v_j)$
    end
    $\bar{L}_T * := \bar{L}_T * \bigcup \pi(w_s)$
  end /* of the for all loop */

  $\bar{L}* := \bigcup_{v_T \in T} \bar{L}_T *$
  $TotalLength := \max_{v_T \in T} \{Length[v_T]\}$

end MinPropLink.
```

We argue that Algorithm *MinPropLink* finds a minimum proper link. Indeed, $\bar{L}*$ is more than a minimum proper link from w_s to T, it is the union of the minimum proper links from w_s and every node v_T of T.

Proposition 10: *The sub-links $\bar{L}_T *$ constructed by Algorithm MinPropLink are proper links. Their union, $\bar{L}*$, is also proper.*

Proof: By Proposition 9, the only way a sub-link $\bar{L}_T *$ might not be proper is if it contains a v-node v_i with more than one predecessor w-node. But whenever a v-node y is added to *PERM,* and to $\bar{L}_T *$, only one w-node, *Predec*[y], is added as well (line " † " in Algorithm *MinPropLink*). Since y is never visited more than once, only one predecessor w-node is associated with it. Therefore $\bar{L}*_T$ is proper. We now prove the second statement. Suppose $\bar{L}*(w_s, v_T)$ and $\bar{L}*(w_s, \bar{v}_T)$ are distinct minimum proper links from w_s to some v_T and $\bar{v}_T$ respectively, where $v_T \neq \bar{v}_T$. If those two sub-links have no v-node in common, then their union must be a proper link. Otherwise, suppose v_j is common to both. The two sub-links, from w_s to v_j, of $\bar{L}*(w_s, v_T)$ and $\bar{L}*(w_s, \bar{v}_T)$ respectively, must be identical because in Algorithm *MinPropLink,* the v-nodes which are possible candidates to be placed in *PERM* are always examined in the same order. Therefore, v_j must have the same predecessor in $\bar{L}*(w_s, v_T)$ as it does in $\bar{L}*(w_s, \bar{v}_T)$. Hence, since the previous argument applies to any node v_j, the union of the two proper links is proper, and by induction, $\bar{L}*$ is proper. •

The proof of minimality is very similar to the proof for Dijkstra's algorithm [MURT 81].

Proposition 11: *A sub-link* $\bar{L}_T*(w_s, v_T)$ *constructed in Algorithm MinPropLink is the shortest link from* w_s *to* v_T*, and Length*$[v_T]$ *is the length of the shortest link.*

Proof: At some stage of the algorithm, assume that the link from w_s to $v_j \in$ *PERM* traced by the labels on the nodes in *PERM* are the shortest links. By the manner in which the labels *Predec* and *Length* are updated in each step, it is clear that for any $v_j \in$ *TEMP* the link from w_s to v_j traced by the current labels is a minimum length link from w_s to v_j among links from w_s to v_j which traverse only through nodes in *PERM.* The length of that link is the distance in the current label *Length*$[v_j]$. Let $y \in$ *TEMP* be the node made permanent next in *MinPropLink.* Let the current distance in the permanent label of y be *Length*$[y]$, and the distance on the current label on $v_j \in$ *TEMP,* $v_j \neq y$ be *Length*$[v_j]$. Then by the choice of y,

$$Length[y] = \min\ \{Length[v_j] : v_j \in TEMP,\ v_j \neq y\}. \qquad \ddagger$$

If the link from w_s to y traced by the current labels is not the shortest, the shortest link from w_s to y must pass through some node $v_j \in$ *TEMP* $v_j \neq y$ before reaching y. Suppose $v_k \in$ *TEMP* is the first such node on this shortest link. Let δ be the length of a shortest link from v_k to y. The length of the shortest link from w_s to y is

$$Length[v_k] + \delta \geq Length[v_k] \geq Length[y],$$

by (‡), which is a contradiction. Therefore the link from w_s to y, traced by the current labels *Predec* is indeed the shortest link from w_s to y. By induction, and the above argument, the assertions in Proposition 11 are valid for the set *PERM* after each stage of the algorithm. •

Note that Algorithm *NearestSource* finds a set of w-nodes *SourceSet* such that, for every $w_s \in SourceSet$, $\pi(w_s) > T$, and for any $v_i \in \pi(w_s)$, $\pi(w_s) - v_i \not> T$, whereas *MinPropLink* explicitly identifies the *shortest* link from a single w-node w_s to all the v-nodes v_T in T. Thus, given a source *Wsource*, *MinPropLink* can be executed for all the nodes $w_s \in Wsource$, and thus the over-all minimum proper link $\bar{L}*(w_s, T)$ can be determined. The computational effort required by Algorithm *MinPropLink* is $O[|V_L|^2 \times |T|]$.

Each node of the so-called metagraph, (**Y**,**L**), consists of a w-node and its predecessors in $\bar{L}*$ if those predecessors contain at least one target node, or if the w-node is the root of $\bar{L}*$, w_s. Additional nodes are accounted for target nodes which do not have successors. Each arc of the metagraph depicts an implicit FD in $\bar{L}*$, induced by a sub-link of $\bar{L}*$. Each such FD contains at least one target attribute on either side of its arrow. Note that, in Algorithm *Metagraph*, since the starting sub-link is proper, all the sub-links are also proper, and therefore Algorithm *GetLink* (more efficient that *MinPropLink*) can be used to determine all the arcs of **L**.

4.4.4.2. Step-3-b-2: Weight assignment on proper links

The size of the FD $f(w_s, T)$, implicitly represented by link $\bar{L}*(w_s, T)$ can easily be calculated after $\bar{L}*$ has been traced back in Algorithm *MinPropLink*. The following algorithm, which estimates the size of $f(w_s, T)$ is very similar to Algorithm *ReachAll* seen earlier, and of the same time complexity $O[|A_v| + |A_w|]$. To minimize estimation errors, the FDs explicitly represented in $\bar{L}*$ are joined from the source w_s until $\bigcup_{v_T \in T} \{Att(v_T)\}$ is covered.

ALGORITHM SizeEstimate: /* Estimates the size of $f(w_s, T)$ implicitly represented by a link $\bar{L}*$ */

```
TargetNodes := $V_{\bar{L}}$
f := $\emptyset$
Size := 1
ReachNodes := $\pi(w_s)$
NextNodes := $\{v : (w_s, v) \in A_{\bar{L}}\}$
for all $w_i \in W_{\bar{L}}$ do
   NoPredec[$w_i$] := $|\pi(w_i)|$
while NextNodes $\neq \emptyset$ & TargetNodes $\neq \emptyset$ do
begin
   Generate := $\emptyset$
   for all $x \in$ NextNodes & $x \notin$ ReachNodes do
   begin
      for all $w_i : (x, w_i) \in A_{\bar{L}}$ do
      begin
         NoPredec[$w_i$] := NoPredec[$w_i$] - 1
         if NoPredec[$w_i$] = 0then do
         begin
            Generate := Generate $\bigcup \{v_k : (w_i, v_k) \in A_{\bar{L}}\}$
            f := f $\bigcup \{f_{ij} : \bigcup_{v \in \pi(w_i)} \{Att(v)\} \rightarrow Att(v_k)\}$
            Size := $\dfrac{Size \times |f_{ij}|}{\prod_{v \in \pi(w_s)} |DOM(Att(v))|}$
         end
      end
   end
   TargetNodes := TargetNodes - NextNodes
   ReachNodes := ReachNodes $\bigcup$ NextNodes
   NextNodes := Generate
end
if

   $PROD \overset{\Delta}{=} \prod_{\left\{\substack{A \in \bigcup_{v \in \pi(w_s)} Att(v) \\ v_T \in T}\right\}} |f_{sj}[A]| \times |f_{iT}[Att(v_T)]| < Size$

                           for some j and i

then Size := PROD
Size($w_s$, T) := Size
```

end SizeEstimate.

In the next section some useful mathematical properties of (EOP) are established. A new model is then proposed to capture the problem of Step-3-b-3. A detailed implementation of

Step-3-b-3 then follows.

4.4.4.3. Step-3-b-3: Matroid model and optimization

In this step, we extract an out-tree of (**Y**,**L**) which is said to *depict* an envelope $K(T)$ in the following sense; $K(T)$ is the envelope that contains the FDs represented by the arcs of the out-tree, and only those FDs. The justifications for this step will be provided later. We will first prove three important properties of (**Y**,**L**). Those properties will later be used to design a very simple and efficient optimization algorithm.

Property-1

Proposition 12: *The metagraph* (**Y**,**L**) *of* $\bar{L}*(w_s,T)$ *does not contain any directed cycle.*

Proof: If there exists a directed cycle in (**Y**,**L**), then there exists a "link-cycle" in $\bar{L}*$ (the notion of proper link in an F-graph is analogous to the notion of directed path in a regular graph, therefore a link-cycle is the obvious extension of a directed cycle). Therefore, either one v-node, in $\bar{L}*$ but not in $\pi(w_s)$, has more than one predecessors, which contradicts Proposition 9, since $\bar{L}*$ is a proper link, or some v-node, y, in $\pi(w_s)$ has a predecessor, z. That latter possibility must be discarded since an arc (z,y) may not be a linking arc from w_s to y with respect to the set of arcs in $\bar{L}*$. •

Property-2

Consider hypergraph $M \overset{\Delta}{=} (\mathbf{L},\Lambda)$ where **L** is, as before, the set of arcs of the metagraph, and Λ is: $\Lambda \overset{\Delta}{=} \{\Lambda_i : \Lambda_i$ is a set of elements of **L**, and Λ_i is an *out-tree** or a *forest* of *out-trees* in (**Y**,**L**) $\} \bigcup \emptyset$. As proved below, M is a *matroid*. (See [LAWL 76]).

Proposition 13: $M = (\mathbf{L},\Lambda)$ *is a matroid, i.e. :*

*An *out-tree* is a directed tree in which each node has at most one arc directed into it.

(i) for any $\Lambda_i \in \Lambda$, *if* $\Lambda_j \subset \Lambda_i$ *then* $\Lambda_j \in \Lambda$. *Also,* $\emptyset \in \Lambda$.

(ii) *If* Λ_i^p *and* $\Lambda_j^{p+1} \in \Lambda$, *contain respectively p and p+1 elements of* $\mathbf{L}$, *then* there exists $L \in \Lambda_j^{p+1} - \Lambda_i^p$ *such that* $\Lambda_i^p \bigcup \{L\} \in \Lambda$.

Proof: (i) The proper subset of an out-tree is an out-tree or a forest of out-trees. Also, by definition, $\emptyset$ is a member of Λ. (ii) Let Λ_i^p be a forest of out-trees with p arcs. Let Λ_j^{p+1} be a forest of out-trees with $p+1$ arcs. We proceed by contradiction. Let us suppose that *for any* $L = (e,f) \in \Lambda_i^p - \Lambda_j^{p+1}$, $\Lambda_i^p \bigcup \{L\}$ is *not* a forest of out-trees. Then, for all $(e,f) \in \Lambda_j^{p+1}$, either both e and f already belong to Λ_i^p, or f is pointed at by an arc in Λ_i^p, whereas e is outside Λ_i^p. (Of course, there must exist at least one arc $(e,f) \in \Lambda_i^p$ such that e is not a node in Λ_i^p). Let us try to reconstruct Λ_j^{p+1}. For each arc (g,h) in Λ_i^p, there exists an arc (e,h) in Λ_j^{p+1} (with e possibly identical to g). Therefore we can account for only p arcs in Λ_j^{p+1}. This contradicts the assumption that Λ_j^{p+1} is a forest of out-trees with $p+1$ arcs. Therefore there must exist at least one $L \in \Lambda_j^{p+1} - \Lambda_i^p$ such that $\Lambda_i^p \bigcup \{L\}$ is an out-tree or a forest of out-trees. Since M satisfies (i) and (ii), therefore it is a matroid. •

All the subsets of $\mathbf{L}$ in Λ are said [LAWL 76] to be *independent sets* of M. A maximal independent set is said to be a *base* of M. By definition of $(\mathbf{Y},\mathbf{L})$ a base is a *spanning out-tree* in $(\mathbf{Y},\mathbf{L})$. **

Property-3

Proposition 14: *Any spanning tree in* $(\mathbf{Y},\mathbf{L})$ *is rooted in* $e_s \overset{\Delta}{=} <\pi(w_s),w_s> \in \mathbf{Y}$.

Proof: This follows from the fact that there exists an arc between e_s and any other node of $\mathbf{Y}$, and from the acyclic property of $(\mathbf{Y},\mathbf{L})$ asserted by Proposition 12. •

It is known that the *matroid greedy algorithm* [LAWL 76] finds a *base of maximum weight*. In this context, it is reasonable to define the weight of an arc

**We know that there is at least one such spanning out-tree by definition of $(\mathbf{Y},\mathbf{L})$.

$L = [<.,w_s> , <V,.>]$ in **L** to be a large number minus the estimated size of the FD $f(w_s, V)$ implicitly represented by link $L(w_s, V)$ associated with L. The general matroid greedy algorithm as described in [LAWL 76] is:

Choose the elements of the matroid in order to size, weightiest element first, rejecting an element only if its selection would destroy independence of the set of chosen elements.

Although the construction of a spanning out-tree is usually not trivial because of potential directed cycles, in this context, a very simple greedy algorithm may be devised for Step-3-b-3:

Assign to each arc in **L** *a large positive number minus the size of the FD implicitly represented by the corresponding link. Choose the elements of* **L** *in order of weight, weightiest element first, rejecting an element only if it points towards the same node of* **Y** *as another already chosen arc.*

That this algorithm indeed finds a maximum-weight spanning out-tree of (**Y**,**L**) follows immediately from Properties 1-3 and the theory of matroids [LAWL 76]. However, we include a comprehensive proof below.

Proposition 15: *The greedy algorithm finds the maximum weight spanning out-tree of* (**Y**,**L**).

Proof: First, let us show that at any step p of the algorithm, the current forest of out-trees Λ_i^p is the weightiest among all other forests of out-trees with the same number of arcs. At step $p = 1$ this is obviously true since the weightiest arc is initially chosen. By induction, suppose Λ_i^{p-1} is the weightiest forest of out-trees with p–1 arcs. Two cases must be considered:

(1) Λ_i^{p-1} can be augmented:

Let $\Lambda_i^{p-1} = \{(e_i, f_i)\ \ i = 1, \cdots, p-1\}$. If the next weightiest arc (g,h) is such that $h = f_m$, for some f_m, then it is not acceptable. Else it is acceptable (by Proposition 12 and the fact that Λ_i^{p-1} is an out-tree), and $\Lambda_i^p = \Lambda_i^{p-1} \bigcup \{(g,h)\}$ is an out-tree and we are done. Suppose that $h = f_m$, for some m. Assume further that there exists an out-tree with p arcs:

$$\Lambda_j^p = \{(\bar{e}_j, \bar{f}_j)\ j = 1, \cdots, p-1\} \bigcup \{(g, f_m)\}$$

such that:

$$\sum_{j=1}^{p-1} Weight(\bar{e}_j, \bar{f}_j) + Weight(g, f_m) > \sum_{i=1}^{p-1} Weight(e_i, f_i) + Weight(k, l)$$

for any acceptable (k,l). Since $Weight(e_1, f_1) > Weight(g, f_m)$ therefore:

$$\sum_{j=2}^{p-1} Weight(\bar{e}_j, \bar{f}_j) + Weight(\bar{e}_1 \bar{f}_1) > \sum_{i=2}^{p-1} Weight(e_i, f_i) + Weight(k, l),$$

which implies that:

$$Weight(\bar{e}_1, \bar{f}_1) > Weight(k, l).$$

But we know that $\bar{f}_1 \neq f_m$ because if this were not true then $\Lambda_j^{p-1} \bigcup \{(g, f_m)\}$ would not be an out-tree. Furthermore, $\Lambda_i^{p-1} \bigcup \{(\bar{e}_1, \bar{f}_1)\}$ is an out-tree (using Proposition 12) weightier than $\Lambda_i^{p-1} \bigcup \{(k,l)\}$. This last statement contradicts the assumption that (k,l) was the next heaviest available arc for Λ_i^{p-1}. Therefore, either an out-tree such as Λ_j^{p-1} does not exist, or Λ_i^{p-1} cannot be augmented further. In the first eventuality the original assertion is true. In the latter, we have to consider case (2):

(2) Λ_i^{p-1} cannot be augmented:

Suppose that there exists an out-tree Λ_j^q with $q > p-1$ such that Λ_j^q is weightier than Λ_i^{p-1}. Therefore, there exists $\Lambda_j^p \subseteq \Lambda_j^q$ such that, by Proposition 13, $L \in \Lambda_j^p - \Lambda_i^{p-1}$ can be added to Λ_i^{p-1} to create a larger out-tree. This contradicts the assumption that Λ_i^{p-1} can no more be augmented. Therefore, Λ_i^p must indeed be the weightiest forest of out-trees at any step p. Since there exists a spanning out-tree of $(\mathbf{Y},\mathbf{L})$, and using the assertion of Proposition 14, the greedy algorithm terminates when the maximum-weight spanning out-tree rooted in e_s is found. ●

Step-3-b-3 may now be justified.

Proposition 16: *An out-tree of* $(\mathbf{Y},\mathbf{L})$ *depicts an envelope* $K(T)$ *and not more.*

Proof: As noted in connection with Theorem 2, there exists a query envelope which is embedded in the link $L(w_s, T)$. It is easy to see that there is also an envelope $K(T)$ embedded in proper sub-link $\bar{L}*$ of L. Therefore, the metagraph (**Y**,**L**) depicts more than is needed for such an envelope $K(T)$. If a subgraph of (**Y**,**L**) is an out-tree, then by Proposition 6, it is an out-tree rooted in $<\pi(w_s),w_s>$. Therefore, the F-graph $\hat{L}$, which has for explicit FDs the FDs associated with the arcs of the out-tree, is a sub-link in the F-graph G^+, where G^+ denotes the F-graph that explicitly represents all the FDs of F^+. By Corollary 2 $\hat{L}$ is lossless. Furthermore, since the out-tree spans all the nodes of **Y**, all the v-nodes of T are covered. Therefore, the out-tree depicts an envelope $K(T)$. The sub-graph extracted from (**Y**,**L**) may be more than an out-tree, but it would then contain extraneous information. An envelope depicted by such a sub-graph of (**Y**,**L**) would conflict with requirement (eop-2). •

The implementation of the greedy algorithm follows:

ALGORITHM OutTree: /* Finds a maximum-weight spanning tree in (**Y**,**L**) rooted in $e_s = <.,w_s>$ */

```
Arborescence := true
OutTree := $\emptyset$
for all $e = [<.,w> , <V,.>] \in \mathbf{L}$
   do $Weight[e] := \mathbf{S} - Size(w,V)$
   /* Size is given by SizeEstimate and S is a large number */
while Arborescence = true do
begin
   $e \overset{\Delta}{=} (e_1,e_2) := ( e_i \in \mathbf{L} :$
         $Weight[e_i] = \max\{Weight[e_k] , e_k \in \mathbf{L} - OutTree\}$
         $and\ \forall\ \bar{e} = (\bar{e}_1,\bar{e}_2) \in OutTree , \bar{e}_2 \neq e_2 )$
   if $e = \emptyset$ then Arborescence := false
   else $OutTree := OutTree \cup \{e\}$
end
```

end OutTree.

Algorithm *OutTree* is of time complexity $O[|\mathbf{L}|^2]$. Note that the total time complexity of Approach 3-b is therefore

$$O\Big[|SourceSet| \times |V_L|^2\Big] + O\Big[|T| \times |V_{\bar{L}*}|^2\Big]$$

$$+\ O\Big[|A_{v_{\bar{L}\bullet}}| + |A_{w_{\bar{L}\bullet}}|\Big] + O\Big[|\mathbf{L}|^2\Big].$$

If a proper link $\bar{L}(w_s, T)$ is given, and if the cost of the model of EOP in Section 4.4.2 is changed to be the sum of the sizes of the FDs in the envelope sought, then Approach 3-b solves exactly the latter model. For most practical cases the expression for the cost given in Section 4.4.2 is unreasonable because of Assumptions EOP-cost-1-2 (page 72). Therefore, a comparison of the solutions obtained respectively by Approach 3-b and an "optimal" algorithm is not justified.

In the next section, some typical problems of FD-updating are considered. The semantic constraints that support the FD-modeling structure control the processing of updates, not unlike join dependencies control the correct processing of queries. Thus, a single update on an individual FD may force, as a side-effect, a so-called cascade of updates on other FDs. The cascade associated with one update is defined to be the set of FDs, which are the target of this update. Section 4.5 addresses the problem of determining update-cascades to preserve database consistency.

4.5. Consistency preservation in updates

When selecting a query envelope, the *lossless join* property was considered to be the main factor to satisfy semantic correctness. Similarly, some criteria must be considered in association with updates of FDs to preserve the integrity of the database. In the next section definitions for simple practical semantic constraints on FDs are provided. These constraints are to help understand how the update of a single FD may lead to a *cascade* of side-effects (Section 4.5.2). In Section 4.5.3 three basic types of updates are considered in details.

4.5.1. Constraints on the antecedent and range of an FD

The notion of update-provoked side-effects will rely, in part, on the following definitions. As explained in Section 3.1, each FD f_k is derived from a function F_k. Using the same notation as in Chapter 3, $A_k \overset{\Delta}{=} DOM(A_1) \times DOM(A_2) \times \cdots \times DOM(A_n)$ is defined to be the *antecedent* of f_k, and $R_k \overset{\Delta}{=} DOM(B_1) \times DOM(B_2) \times \cdots \times DOM(B_m)$ is defined to be the *range* of f_k. For some $x \in A_k$ and $y \in R_k$, the notation $y = f_k(x)$ will be used.

(1) f_k is *partial* with respect to A_k if there exists $x \in A_k$ such that x has no corresponding image in R_k with respect to f_k. We write $f_k(x) = \emptyset$.

(2) f_k is *total* with respect to A_k if for any $x \in A_k$, there exists $y \in R_k$ such that $f_k(x) = y$.

(3) f_k is *partial* with respect to R_k if there exists $y \in R_k$ such that for any $x \in A_k$, $f_k(x) \neq y$.

(4) f_k is *total* with respect to R_k if for any $y \in R_k$, there exists $x \in A_k$ such that $f_k(x) = y$.

On an F-graph, the tail and the head of each w-arcs are labelled with either the letter P (partial) or T (total) as shown on Figure 4.7. The need for maintaining such labels for each FD is illustrated in the following example from [HOUS 79].

Example: A nation-wide organization keeps records of their MEMBERs and of their SPOUSE records if they are married. The MEMBER and SPOUSE attributes are related by the FD f_{MS} which stands for $MEMBER \rightarrow SPOUSE$. Those MEMBERs who are not married will not be part of FD f_{MS}. Therefore f_{MS} is *partial* with respect to its *antecedent* MEMBER, and is *total* with respect to its *range* SPOUSE. Only the SPOUSEs of known MEMBERS in the organization appear in the database. We write $MEMBER(P) \rightarrow SPOUSE(T)$. •

4.5.2. Deletion of attribute values

It is obvious that the deletion of an attribute value with respect to an FD can trigger a *cascade* of deletions on related FDs. This "side-effect" phenomenon has been studied previously in [HOUS 79] and is addressed in Section 4.5.3. Typically, it is important to keep track of the labels of any FD under updating. This is particularly true in the case of deletions of attribute values or of FD tuples. The following example is to help understand why a single deletion may require further deletions in *cascade*.

Example: (a) Let FD f_i which stands for $MEMBER(P) \rightarrow SPOUSE(T)$ be represented by the following table:

f_1

MEMBER	SPOUSE
M1	S1
M2	S2
M3	S3
M4	S4

If M1 is to be deleted from the database, then there is no use in keeping S1 around in the database. If S1 is deleted then some other information related to S1 through other FDs may have to be deleted too.

(b) Consider f_2 which stands for $MEMBER\_MICH(T) \rightarrow US\_CENTER(P)$ which relates the members belonging to the organization in the state of Michigan, and all the centers that the organization possesses all over the US. Every members in Michigan relate to some CENTER, thus the label "T" assigned to MEMBER_MICH, but some other centers in the US are obviously not present in f_2, thus the label "P" assigned to US_CENTER in f_2. f_2 is represented by the following table.

f_2

MEMBER_MICH	US_CENTER
M1	C1
M2	C1
M3	C2
M4	C2

If center C1 is closed, then all the members assigned to that center are either re-assigned, in which case the deletion is immediately followed by an update to write the new CENTER information, or they are released, in which case M1 and M2, are automatically deleted from the database. Some simple side-effects of deletions have thus be illustrated. ●

In the next section another source of update cascades will be mentioned. It will then be shown how to handle typical updates and how to identify the FDs affected by a cascade of updates triggered by a single update on attribute values or a tuple of an FD*.

4.5.3. Update cascades

Typical update side-effects are now indicated. Let H be the union of all the query envelopes. If it happens that $H^+ = F^+$, where F is the user-provided set of FDs, then we are done. Else, if $H^+ \subset F^+$, then $(H \bigcup I)^+ = F^+$, where $I \overset{\Delta}{=} F^+ - H^+$. H^+ and F^+ can easily be derived by applying repeatedly the FD rules of Chapter 3 on H and F respectively. An LR-minimum cover of I, $\bar{I}$ can be derived using [MAIE 80]'s algorithm or Algorithm *Reduce* (Section 4.2.1). Let $\bar{G}$ be the final F-graph which depicts $H \bigcup \bar{I}$. From now-on, it will be assumed that any update is specified with respect to $\bar{G}$, and that all "P" and "T" labels are specified with respect with the explicit FDs of $\bar{G}$.

*It is beyond the scope of this thesis, however, to address all the intricacies associated with updating. This section only considers the most simple cases of updates. Obviously, the more complex the modeling of *user views* is, the more delicate the update problems become.

The typical updates that are considered are:

(a) *Add a tuple* in the instance of FD $f(w,v)$ for some arc (w,v) in $\overline{G}$. If $\bigcup_{v_i \in \pi(w)} \{Att(v_i)\} = \{A_1, \cdots, A_n\}$, and $Att(v) = B$, then the new inserted tuple is taken from $DOM(A_1) \times \cdots \times DOM(A_n) \times DOM(B)$. This type of update may inherently imply the addition of a value in the antecedent of f, or in its image, or both.

(b) *Remove a tuple* in the instance of FD $f(w,v)$ for some arc (w,v) in $\overline{G}$.

(c) *Remove a value* under $Att(v)$ for some v, or under $\bigcup_{v_i \in \pi(w)} \{Att(v_i)\}$ for some w in $\overline{G}$.

It was shown earlier that the semantic constraints represented by the "P" and "T" labels could force a single deletion update of type (b) or (c) to trigger off some further updates in cascade. Similarly, the first universal relation assumption (Assumption UR-1 of Chapter 3), must always be enforced whenever an update of type (a) is made. This is a consideration in the design stage, as well as in the operating stage. In the design stage, enforcing Assumption UR-1 serves to identify the set of FDs affected by a cascade. In the operating stage, it serves to define some actual data processing. Typically, the updates occurring in the operating stage are predictable, and therefore, the designer is able to determine in advance what the specific side-effects are. For non-scheduled or ad-hoc updating transactions, the tedious task of enforcing the semantic constraints* must take place under the careful supervision of the database administrator. Obviously, if an updating transaction consists in a conditional update, i.e. a potential update preceded by a question, then the "question" part of the transaction may be analyzed individually with other regular queries. Each type of update is considered below.

*In particular the identification of all the specific updating operations implied by one update of type (a).

4.5.3.1. Adding a tuple in the instance of an FD

Let $f(w_i,v_i)$ stand for $\{\bigcup_{v_k \in \pi(w_i)} \{Att(v_k)\} \rightarrow Att(v_i)\} = A_{i_1}A_{i_2} \cdots A_{i_n} \rightarrow A_{i_m}$. A new tuple of $f(w_i,v_i)$ is added. It is of the form $d_i = (d_{i_1}, \cdots, d_{i_n}, d_{i_m})$, where $d_{i_k} \in DOM(A_{i_k})$. This type of update is the most laborious because of the potential disuption of the "uniqueness" assumption (Assumption UR-1, Chapter 3). More specifically, as a new tuple of $f(w_i,v_i)$ is added, the uniqueness assumption may be endangered whenever $f(w_i,v_i)$ belongs at least to one *doublet*. The principle behind the correction of any trouble-shooting is rather simple; it may be stated as follows:

Let (w_i,v_i) be the arc that corresponds to FD $f(w_i,v_i)$ for which a new tuple is to be added. For each doublet $(\bar{L}_j^+,\bar{L}_j^-)$ each ending at node v_t, to which the arc (w_i,v_i) belongs, check if for the tuple d_i which is added in $\bar{L}_j^+$, the image d_t in $DOM(Att(v_t))$ is the same along $\bar{L}_j^+$ and $\bar{L}_j^-$, and the inverse image relation, say r_s, is the same along $\bar{L}_j^+$ and $\bar{L}_j^-$. If not, the update is not correct. If d_i has an image d_t and an inverse image r_s, augment $\bar{L}_j^-$ such that the image of r_s in $DOM(Att(v_t))$ along $\bar{L}_j^-$ is d_t too.

Using the notation of Chapter 3, the translation of this action becomes quite laborious. It appears below after some notational definitions.

Let $\Gamma \overset{\Delta}{=} (C,\bar{C})$ denote a cutset of proper link $\bar{L}_j^+(w_s,v_t)$ which itself belongs to a doublet $(\bar{L}_j^+,\bar{L}_j^-)$. We write:

$$X_C \overset{\Delta}{=} \bigcup_{\substack{v \in \pi(w) \\ w \in C}} \{Att(v)\}$$

$$X_{\bar{C}} \overset{\Delta}{=} \bigcup_{v \in \bar{C}} \{Att(v)\}$$

$$X_\Gamma \overset{\Delta}{=} X_C \bigcup X_{\bar{C}}$$

$$X_s \overset{\Delta}{=} \bigcup_{v_s \in \pi(w_s)} \{Att(v_s)\}$$

$$X(\bar{L}_j^{+}) \overset{\Delta}{=} \bigcup_{v \in V_{\bar{L}_j^{+}}} \{Att(v)\}$$

$$E(\bar{L}_j^{+}) \overset{\Delta}{=} \bigcup_{(w,v) \in A_{W_{\bar{L}_j^{+}}}} f(w,v).$$

In the following algorithm, the notation $I_{<X,>}$ denotes the largest relation in $I_{<X,>}$.

ALGORITHM AddTuple: /* Add a tuple of $f(w_i, v_i)$ */

```
$ALTER := \{\Gamma_i\}$
$Current := ALTER$ /* Cutsets still to be visited */
/* $\Gamma_L = (w_i, v_i)$ if $(w_i, v_i)$ is a cutset
  of $\bar{L}_j^{+}$, otherwise it is any cutset of $\bar{L}_j^{+}$
  which contains $(w_i, v_i)$. */
for all $\Gamma_i \in Current$ while $Current \neq \emptyset$ do
begin
   $Current := Current - \{\Gamma_i\}$
   for all $(\bar{L}_j^{+}, \bar{L}_j^{-}) : \bar{L}_j^{+}$ contains $\Gamma_i$ do
   begin
      /* Trace forward in $\bar{L}_j^{+}$ */
   $d_{t_i} := \left\{ I_{<X(\bar{L}_j^{+}), E(\bar{L}_j^{+})>}\left[ X_{\Gamma_i} \bigcup \{Att(v_t)\},\ t(X_{\Gamma_i}) = d_i \right] \right\}[Att(v_t)]$
      if $d_{t_i} = \emptyset$ then done
      else begin
         /* Trace backwards in $\bar{L}_j^{+}$ */
   $r_{s_i} := \left\{ I_{<X(\bar{L}_j^{+}), E(\bar{L}_j^{+})>}\left[ X_s \bigcup X_{C_i},\ t(X_{C_i}) = d_i(X_{C_i}) \right] \right\}[X_s]$
         if $r_{s_i} = \emptyset$ then done
         /* Now trace back from $v_t$ to $w_s$ on $\bar{L}_j^{-}$ */
         $k := 0$ /* An index to identify cutsets on $\bar{L}_j^{-}$ */
         $d_k := d_{t_i}$
         $r[k] := \{d_k\}$
         $X_{\bar{k}} := Att(v_t)$
         $X_k := \{Att(v) : v \in \pi(w)$
         & $w \in \pi(v_t)\} \bigcap \bar{L}_j^{-}$
         while $X_k \neq X_s$ & $r[k] \neq \emptyset$ do
         begin
            $k := k + 1$ /* Next cutset to visit */
   $r[k] := \left\{ I_{<X(\bar{L}_j^{-}), E(\bar{L}_j^{-})>}\left[ X_k \bigcup X_{\bar{k}},\ X_{\bar{k}} = d_k\ \&\ d_k \in r[k] \right] \right\}[X_k]$
            $X_{\overline{k+1}} := X_k$
            $V_{\overline{k+1}} := \emptyset$
            for all $v_i : Att(v_i) \in X_{\overline{k+1}}$
```

```
        do $V_{\overline{k+1}} := V_{\overline{k+1}} \bigcup \{v_i\}$
        $X_{k+1} := \{Att(v) : v \in \pi(w)$
        $\& \; w \in \pi(V_{\overline{k+1}})\} \bigcap \bar{L}_j^-$
    end
    if $r_{s_i} \in r[k]$ then done
    else begin /* No antecedent for some cutset */
        /* $k = Length(\bar{L}_j^-)$ */
        $s[k] := r_{s_i}$
        $X_k := X_s$
        $X_{\bar{k}} := \{Att(v) : v \in \sigma(w_s) \bigcap \bar{L}_j^-\}$
        while $X_k \neq Att(v_t)$ do
        begin
            $k := k - 1$
$s[k] := \left\{ I_{<X(\bar{L}_j^-),E(\bar{L}_j^-)>}\left[ X_k \bigcup X_{\bar{k}},\, X_k = d_k \;\&\; d_k \in s[k] \right] \right\}[X_{\bar{k}}]$
            for all $A_k \in X_{\bar{k}}$ do
            begin
                if $(s[k])[A_k] = \emptyset$
                then begin
                    $q[k] := r[k] - s[k]$
                    modify $(w,v) : Att(v) = A_k$
                    such that each tuple in $s[k]$
                    which features a $\emptyset$ value is
                    mapped to a tuple in $q[k]$.
                    /* ... User controlled */
                    $ALTER := ALTER \bigcup \{\Gamma : (w,v) \in \Gamma\}$
                    $Current := Current \bigcup \{\Gamma : (w,v) \in \Gamma\}$
                end
            end
        end
    end
  end
 end
end

end AddTuple.
```

Note that all the proper links may be determined with Algorithm *Distinguish* (section 4.4.3.1), as soon as the modified F-graph depicting H is completed.

The side-effects due to a *deletion* have been mentioned in [HOUS 79]. Typically, they take place whenever an affected FD does not feature a "P" label for both its antecedent and its image. Again, the translation of simple deletion cascades into pseudocode is quite lengthy. The next two sections deal with deletion.

4.5.3.2. Removing a tuple in the instance of an FD

A tuple $d_i = (d_{i_1}, d_{i_2}, \cdots, d_{i_n}, d_{i_m})$ is to be deleted from the database. The following steps are taken:

ALGORITHM DeleteTuple: /* Remove a tuple of $f(w_i, v_i)$ */

```
    $ALTER := \{(w_i, v_i)\}$
    if $LABEL[(w_i, v_i)] = (T, T)$
    then begin
       $DeleteValue(< d_{i_1}, \cdots, d_{i_n} >)$
       if $\forall\ t \in f(w_i, v_i)\ t(Att(v_i)) \neq d_{i_m}$
       then $DeleteValue(d_{i_m})$
    end
    else if $LABEL[(w_i, v_i)] = (T, P)$
       then $DeleteValue(< d_{i_1}, \cdots, d_{i_n} >)$
       else if $LABEL[(w_i, v_i)] = (P, T)$
          then begin
             if $\forall\ t \in f(w_i, v_i)$ , $t(Att(v_i)) \neq d_{i_m}$
             then $DeleteValue(d_{i_m})$
          end

end DeleteTuple.
```

4.5.3.3. Removing a value in the antecedent or range of an FD

ALGORITHM DeleteValue: /* Given $d \overset{\Delta}{=} < d_{i_1}, \cdots, d_{i_n} > \in DOM(A_1) \times \cdots \times DOM(A_n)$ and $ALTER$ possibly empty */

```
    $Possible := \emptyset$ /* Possible set of FDs in a cascade */
    for all $f(w_j, v_j) : \{A_1, \cdots, A_n\} = \bigcup_{v \in \pi(w_j)} \{Att(v)\}$ do
    begin /* Delete an image value */
       $Possible := Possible \bigcup \{f(w_j, v_j)\}$
       if ($LABEL[(w_j, v_j)] = (T, T)$ or
          $LABEL[(w_j, v_j)] = (P, T)$) then
       begin
          if $\forall\ t \in f(w_k, v_j)$
            for some $k$ : $t(Att(v_j)) \neq d_{i_m}$
            where $d_{i_m}$ is the image of $d$ in $f(w_j, v_j)$
           then do
              $DeleteValue(d_{i_m})$
              $ALTER := ALTER \bigcup \{(w_j, v_j)\}$
            end
        end
    end
    for all $f(w_j, v_j) \in Possible$ do
    begin /* Delete an antecedent value */
        if ($LABEL[(w_j, v_j)] = (T,T)$ or
            $LABEL[(w_j, v_j)] = (T,P)$) then
        begin
            for all ($f(w_j, v_j) \in Possible$
                : $t[A_1, \cdots, A_n] = d$) do
            begin
                $ALTER := ALTER \bigcup \{(w_j, v_j)\}$
                $DeleteValue(d)$
            end
        end
    end

end DeleteValue.
```

For any update, all the FDs in *H* which are affected by a cascade are the FDs represented by the elements of set *ALTER*.

In the next section the problem addressed in this chapter is summarized together with the proposed solution methodology.

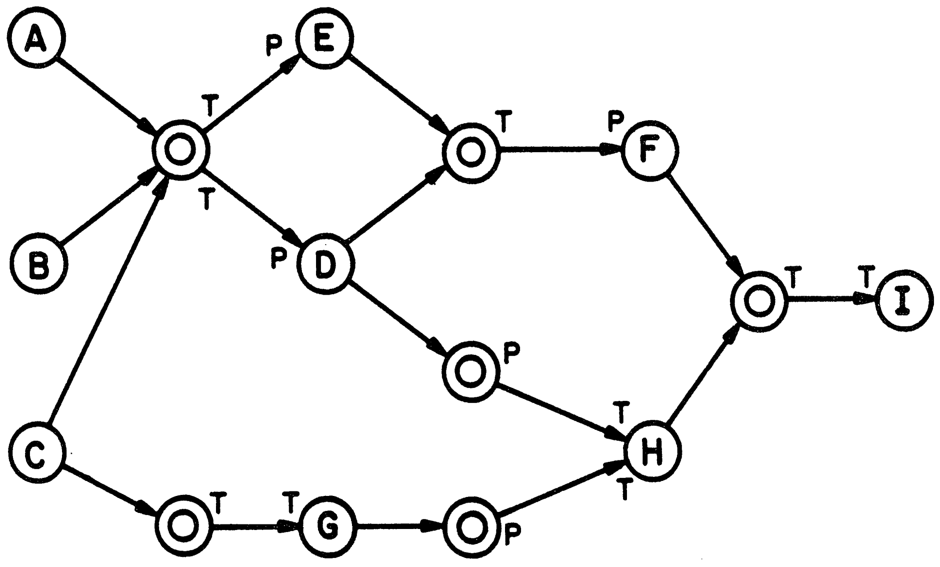


Figure 4.7. An F-graph with "partial" and "total" labels

4.6. Summary of PART 1: From users' FDs to envelopes

The design methodology of PART 1 for the distributed database design problem (as defined in Section 3.3) is now summarized in the form of a design algorithm named *Selection.*

Each instruction in the forthcoming Algorithm *Selection* is numbered for easy referencing, and is of the form:

(*i*) *Output list* ⟵ *Algorithm*[+*Algorithm* · · ·][*Input list*],

where "(i)" is the reference index. The items in an input or output list are separated by commas. Symbols preceded by a double dagger sign "‡" are understood to be user-provided inputs. If several algorithms are used, they are separated by plus signs "+". Most of the algorithms have been defined in pseudocode in the course of Chapter 4. Some other algorithms will simply be defined in words because they involve actions which can be trivially realized. Symbols which have not yet been defined will be defined following the presentation of *Selection* or will be defined in Chapter 5. Algorithm *Selection* is shown below and is followed by specific remarks for each instruction.

```
DESIGN-Selection: /* PART 1 of the distributed database design */

    begin /* Step-1: Initial processing of the user F-graph */
      (0)   Given ‡ $F$ and ‡ $U$, construct $G_F$
      (1)   $\overline{G}_F$ ⟵ Reduce+ReachAll[$G_F$]
      (2)   $\overline{G}_F *$ ⟵ NoLoss[$\overline{G}_F$]
    end

    for all $Q_{tk}$ do
    begin
      /* Step-2: Extract the minimum information */
      (3)   Source ⟵ GetSources[$\overline{G}_F *$, ‡ $T_{tk}$]
      (4)   NearestSource ⟵ GetNearestSource[$\overline{G}_F *$, $T_{tk}$, Source]
      for all $w_s \in$ NearestSource do
        (5)   $L(w_s, T)$ ⟵ GetLink[$\overline{G}_F *$, $T_{tk}$, NearestSource]

      /* Step-3: Find envelope */
      (6)   $\Omega_{tk}$ ⟵ LocalProc[ ‡ $\Omega$, ‡ $C_{tk}$]
      for all $w_s \in$ NearestSource do
        (7)   $\overline{L}*(w_s, T)$ ⟵ MinPropLink[$L(w_s, T)$]
    $w_s$ : $TotalLength(\bar{L}*(w_s,T)) = \min \{TotalLength(\bar{L}*(w_i,T))$
    V $w_i \in NearestSource\}$
    (8)    $(\mathbf{Y},\mathbf{L}) \longleftarrow Metagraph[L(w_s,T)]$
    /* ... Note that the original link is used, $L$ rather than $\bar{L}*$ */
    for all $L = [(.,w);(V,.)] \in \mathbf{L}$ do
       (9)    $Size(w,V) \longleftarrow SizeEstimate[(\mathbf{Y},\mathbf{L}), \Omega_{tk}, \Omega]$
    (10)    $ACC_{tk} \longleftarrow OutTree[(\mathbf{Y},\mathbf{L}),$ V $L \in \mathbf{L}$ $Size(L)]$
    (11)    $PAR_{tk} \longleftarrow AllPartitionsOf[ACC_{tk}]$
    (12)    $CLU_{t}^{p}{}_{k} \longleftarrow AllClustersOf[p \in PAR_{tk}]$
  end
  (13)    $H \longleftarrow \bigcup_{Q_{tk}} [ACC_{tk}]$
  Construct $G_H$ ,the F-graph of H

  for all $Q_{tk}$ do
  begin /* Step-4: Find distributed query processing data-volumes */
    for all  $p \in PAR_{tk}$ do
    begin
       (14)    $query\ {}_{t}^{p}{}_{k} \longleftarrow QuerySpec[\Omega,$ V $L \in \mathbf{L}, Size(L),$
                    $C_{tk},\ \ddagger \nu_{tk}, ACC_{tk}, p \in PAR_{tk}, CLU_{t}^{p}{}_{k}]$
       V $q,r \in CLU_{t}^{p}{}_{k}$
       (15)    $A_{t}^{p}{}_{k}^{q}, A_{t}^{p}{}_{k}^{qr}, \bar{A}_{t}^{p}{}_{k}^{qr} \longleftarrow Algorithm\ H\ [CHUN83][query\ {}_{t}^{p}{}_{k}]$
    end
  end

  for all $U_{sk}$ do
  begin /* Step-3 Find cascade */
    (16)    $DEL_{sk} \longleftarrow UpdateSpec[G_H, T_{sk}]$
    (17)    $WRT_{sk}$,  V $h \in WRT_{sk}$ $B_{s}^{h}{}_{k} \longleftarrow AddTuple$
                                  $+DeleteTuple$
                                  $+DeleteValue$
                                  $+Distinguish$
                                  $[G_H,$
                                  $\Omega,$
                                   $\ddagger \mu,$
                                  $\nu_{sk}, \ddagger\ TYPE\ , DEL_{sk}]$
  end

end Selection.
```

The following comments apply respectively to every instructions in *Selection*:

(0) F and U are respectively the user-provided set of FDs and universe of attributes. G_F is the F-graph which depicts the FDs of F.

(1) Algorithm *Reduce* can be found in Section 4.2.1. Algorithm *ReachAll* has been seen in Section 4.3. Note that *ReachAll* is used to determine the set *Generate* of reachable v-nodes, referred to as V_i in *CONDITION* Ω_{3-4} in *Reduce*.

(2) Algorithm *NoLoss* can be found in Section 4.2.3. It includes a covering algorithm such as the one reported in [ETCH 77]. $\overline{G_F}$ * is a lossless fully reduced F-graph.

(3) T_{tk} is defined in Section 3.2. Algorithm *GetSources* is shown in Section 4.3. It calls Algorithm *ReachAll*.

(4) See Section 4.3.

(5) T stands for the set of target nodes which have an index equal to one in T_{tk}. Algorithm *GetLink* can be found in Section 4.3.

(6) Ω is a set of user-provided statistical informations and is defined as:

$$\Omega \overset{\Delta}{=} \{\ |f|, \bar{f}, |f[A]|, \bar{A}, |DOM(A)|, \textit{for all } f \in F \ \& \ A \in U\ \}.$$

All the notations have been defined in Section 4.4.2. Any query Q_{tk} is expressed in a query language with respect to the universal relation scheme $<U, >$. A query specification is of the form:

$$Q_{tk} : \ FIND \ (A_1, \cdots, A_m)$$

$$C_{tk} \left\{ \begin{array}{l} [WHERE\ (A_{i_1} = c_{i_1})] \\ \quad [AND\ (A_{i_2} = c_{i_2})] \\ \qquad \vdots \\ \quad [AND\ (A_{i_n} = c_{i_n})] \end{array} \right.$$

where $A_i \in U$ and $c_i \in DOM(A_i)$. The square brackets enclose an optional qualification clause. All the attributes mentioned have an index equal to one in T_{tk}. C_{tk} is the set of qualification clauses for query Q_{tk}. Ω_{tk} is the statistical information for the universal relation obtained from Ω after initial processing of the FDs in F in order to satisfy the qualification clauses of C_{tk}. Thus, *LocalProc* denotes the action of processing the initial FDs of F by eliminating unwanted tuples.

$$\Omega_{tk} \overset{\Delta}{=} \{ \, |f_{tk}|, \bar{f}_{tk}, |f_{tk}[A]|, \bar{A}, \; for\ all \;\; f \in F \; \& \; A \in U \, \}$$

where f_{tk} is f processed under C_{tk} using the *selection* relational operation if it is possible to do so.

(7) Algorithm *MinPropLink* can be found in Section 4.4.4.1. In this algorithm *TotalLength* denotes the length of $\bar{L}*(w_s, T)$.

(8) Algorithm *Metagraph* can be found in Section 4.4.4.

(9) $Size(w, V)$ is the same as $Size(w_s, T)$ in Algorithm *SizeEstimate* in Section 4.4.4.1, where w_s would be equal to w and T would be equal to V.

(10) ACC_{tk} (for "access") stands for the set of FDs in the envelope assigned to query Q_{tk}, where the notation will be carried on in Chapters 5 and 6.

(11) This step and the next one will become clear in Chapter 5, where the sets PAR_{tk} and CLU_{tk}^{p} are defined. In short, PAR_{tk} is a set of indexes where each index corresponds to a different partition of ACC_{tk}.

(12) See Chapter 5. CLU_{tk}^{p} is the partition of ACC_{tk} which corresponds to index p in PAR_{tk}. Each element of CLU_{tk}^{p} is called a *cluster* and may be visualized as one single relation.

(13) Self explanatory.

(14) Given a partition CLU_{tk}^{p} of ACC_{tk}, let:

$$CLU_{tk}^{p} \overset{\Delta}{=} \{q_1, \cdots, q_m\} \equiv \{R_1, \cdots, R_m\},$$

where each cluster q_i is mapped to one relation R_i. Each relation R_i denotes a join of FDs in $q_i \subseteq ACC_{tk}$, i.e.: $R_i = \coprod_{j=1}^{l} f_j$ for some cluster $q_i = \{f_1, \cdots, f_l\} \subseteq ACC_{tk}$, $\bigcup_{i=1}^{m} q_i = ACC_{tk}$ and $q_i \bigcap q_j = \emptyset$ $for\ i \neq j \in \{1, \cdots, m\}$. The input list includes $Size(L)$ for any $L \in \mathbf{L}$. Therefore the size of each FD in ACC_{tk} is known. Hence, the size of each relation R_i may be computed using the technique of Section 4.4.1. Assuming that relations $R_1, \cdots, R_m$ are located at different sites of a computer network, query Q_{tk} may be processed as a distributed query using [CHUN 83]'s *Algorithm H*. *query* ℓ^p_k denotes the input file of *Algorithm H* [CHUN 83] which represents query Q_{tk} to be processed as a distributed query on distinct relations $R_1, \cdots, R_m$.

(15) The symbols $A\ell^{pq}_k$, $A\ell^{pq'}_k$ and $\bar{A}\ell^{pq'}_k$ are data-volume rates and are defined in Chapter 5. All data-volumes take into account the frequency ν_{tk} of the query.

(16) For an update U_{sk}, *UpdateSpec* refers to the identification of the FDs in H which possess one or more attributes with an index of one in T_{sk}.

(17) The TYPE is either "ADD" or "DELETE". The volume of one update message is assumed to be μ and the volume of one piece of data needed for the update of one FD h is indicated in algorithms *AddTuple, DeleteTuple,* and *DeleteValue* in Section 4.5. Algorithm *Distinguish* is shown in Section 4.4.3.1. WRT_{sk} denotes all the FDs updated in cascade by U_{sk} as given by set *ALTER* in *AddTuple, DeleteTuple* and *DeleteValue*. B^h_{sk} denotes the data and message volume required to update $h \in WRT_{sk}$.

Each FD in H can be mapped to a CRDU. The next chapter will address PART 2 of the distributed database design problem; the distribution of CRDUs. This chapter is closed with an example.

4.7. Example

Consider the MRDB (medical record database) of Appendix C. The set of user-provided FDs is shown in Table C.1. It is shown below how to apply the design procedure *Select* for MRDB. Only queries $Q_{t,6}$ and $Q_{t,7}$ are considered. Design procedure *Select* is stepped through instruction by instruction:

(0) F is the user-provided set of FDs shown in Table C.1. G_F is the F-graph which depicts F (Figure 4.8). The set of attributes U is represented by the v-nodes in G_F.

(1) Applying *Reduce* on G_F:

• By RULE 1 w-nodes w_1, w_2, w_3, w_4, w_5, and w_6 are combined into one w-node w_1. Similarly, w-nodes w_{12} and w_{14} are combined into w_{12}.

• RULE 2 needs not be applied.

• By RULE 3, w_{13}, (v_d, w_{13}), (v_i, w_{13}) and (w_{13}, v_j) are deleted.

• By RULE 4 (see Figure 4.1-ii) all transitive redundancies are eliminated. Therefore, (w_1, v_g), (v_b, w_9), w_9, (w_9, v_k) and (w_1, v_i) are deleted. Also by RULE 4, arc (v_i, w_{11}) is deleted (Figure 4.2). Finally, and using the notation employed in RULE 4: Let $v_T = v_g$, $w_i = w_7$, $w_j = w_{11}$. Also, $V_i = \{v_a\}$, $V_j = \{v_c, v_d, v_e\}$, $\pi(w_i) = \{v_a, v_i\}$ and $\pi(w_j) = \{v_c, v_d, v_e\}$. Hence, $\overline{V_i} = \{v_i\}$ and $\overline{V_j} = \emptyset$. Therefore, are deleted: $(w_i, v_T) = (w_7, v_g)$, w_7, (v_i, w_7) and (v_a, w_7). See Figure 4.3. At this step the fully reduced F-graph is $\overline{G_F}$ displayed in Figure 4.9.

(2) Since v-node v_a (labelled with attribute ORDER) is such that: For any $v \in V$, $v_a > v$, therefore $\overline{G_F}* = \overline{G_F}$, i.e. the graph is lossless and there is no need to apply *NoLoss*.

(3) Let query $Q_{t,5}$ be expressed as:

$Q_{t,5}$: FIND (*ORDER, PATIENT_NO, NAME*)

$C_{t,5}$: WHERE *WARD* = ' *Outpatient* ' .

The target list of query $Q_{t,5}$ is {*ORDER, PATIENT_NO, NAME, WARD*}. The corresponding set of target nodes is $T = \{v_a, v_d, v_i, v_k\}$. By applying Algorithm *GetSource*, it is found that $Source = \{w_1\}$.

(4) By applying Algorithm *GetNearestSource*, it is found that $NearestSource = \{w_1\}$, i.e. $w_s = w_1$.

(5) The link found by *GetLink* for $Q_{t,5}$ is:

$L(w_s, T) = L(w_1, \{v_a, v_d, v_i, v_k\})$. Link $L(w_s, T)$ is depicted in Figure 4.10.

(6) The link found in (5) for query $Q_{t,5}$ depicts the following FDs:

$f_{1,b}$: *ORDER* → *SERVICE*

$f_{1,d}$: *ORDER* → *PATIENT_NO*

$f_{8,f}$: *SERVICE* → *PHYSICIAN*

$f_{12,i}$: *PATIENT_NO* → *NAME*

$f_{10,k}$: *PHYSICIAN* → *WARD*

$f_{15,k}$: *NAME* → *WARD.*

The statistical informations Ω are contained in Tables C.4, C.5, C.6 and C.7.

When the base FDs of $\overline{G}_F$, and in particular the FDs depicted in $L(w_s, T)$ are restricted by qualification clause $C_{t,5}$ (*WARD* = ' *Outpatient*') the statistics are modi-

FD	$\lvert f \rvert$ *Tuples*	$\overline{f}$ *Bytes*
$f_{1,b}$	5075	9
$f_{1,d}$	5075	8
$f_{8,f}$	49	20
$f_{12,i}$	84	19
$f_{10,k}$	23	20
$f_{15,k}$	84	20

Table 4.1. Cardinalities and widths of the FDs in the envelope of query $Q_{t,5}$."

fied and become those of Tables 4.1, and 4.2 (where only the base-FDs of $L(w_s, T)$ are kept). The process of calculating or estimating this new set of statistical informations $\Omega_{t,5}$ has been referred to as *LocalProc* in *Select*. Typically, *LocalProc* can consist in actually performing the relational algebra operation "Selection" implied by $C_{t,5}$ directly on a centralized version of the MRDB database, or it can consist in multiplying the original statistics by an estimated user-provided ratio associated with $C_{t,5}$. Appart from $|DOM(WARD)| = 1$, the remaining statistics on widths of attributes and cardinalities of domains are not affected by *LocalProc*.

(7) There is only one possible source node $w_s = w_1$, therefore this step leads to the proper link $\bar{L}*(w_s, T)$ of Figure 4.11.

FD	ORDER	SERVICE	PATIENT_NO	PHYSICIAN	NAME	WARD
$f_{1,b}$	140	48				
$f_{1,d}$	140		84			
$f_{8,f}$		48		20		
$f_{12,i}$			84		84	
$f_{10,k}$				3		1
$f_{15,k}$					12	1

Table 4.2. Cardinalities of FDs, in the envelope of $Q_{t,5}$, projected on each referenced attribute.

(8) Applying Algorithm *Metagraph* and *MinPropLink* on $L(w_s, T)$, the metagraph of Figure 4.12 is obtained.

For this metagraph, the arcs are defined as follows:

$L_{s1} \overset{\Delta}{=} [<v_a, w_1> \; ; \; <v_d, w_{12}>]$

$\equiv \{v_a, (v_a, w_1), w_1, (w_1, v_d), v_d\}$

$L_{12} \overset{\Delta}{=} [<v_d, w_{12}> \; ; \; <v_i, w_{15}>]$

$\equiv \{v_d, (v_d, w_{12}), w_{12}, (w_{12}, v_i), v_i\}$

$L_{23} \overset{\Delta}{=} [<v_i, w_{15}> \; ; \; <v_k, \emptyset>]$

$\equiv \{v_i, (v_i, w_{15}), w_{15}, (w_{15}, v_k), v_k\}$

$L_{s2} \overset{\Delta}{=} [<v_a, w_1> \; ; \; <v_i, w_{15}>]$

$\equiv \{v_a, (v_a, w_1), w_1, (w_1, v_d), v_d, (v_d, w_{12}), w_{12}, (w_{12}, v_i), v_i\}$

$L_{s3} \overset{\Delta}{=} [<v_a, w_1> \; ; \; <v_k, \emptyset>]$

$\equiv \bar{L}*(w_s, T)$

$L_{13} \overset{\Delta}{=} [<v_d, w_{12}> \; ; \; <v_k, \emptyset>]$

$\equiv \{v_d, (v_d, w_{12}), w_{12}, (w_{12}, v_i), v_i, (v_i, w_{15}), w_{15}, (w_{15}, v_k), v_k\}$.

Further, the arcs in **L** are associated with implicit or explicit FDs as follows:

$L_{s1} \sim f_{1,d}$: *ORDER* → *PATIENT_NO*

$L_{12} \sim f_{12,i}$: *PATIENT_NO* → *NAME*

$L_{23} \sim f_{15,k}$: *NAME* → *WARD*

$L_{s2} \sim f_{1,i}$: *ORDER* → *NAME*

$L_{s3} \sim f_{1,k}$: *ORDER* → *WARD*

L_{13} ˜ $f_{12,k}$: *PATIENT_NO* → *WARD.*

(9) It is understood that all FDs have been restricted by *LocalProc* by virtue of qualification clause $C_{t,5}$. The sizes of the FDs in (8) are calculated by *SizeEstimate* as follows:

$Size(w_1, v_d) = 5075 \times 8 = 40{,}600$ *bytes.*

$Size(w_{12}, v_i) = 84 \times 19 = 1{,}596$ *bytes.*

$Size(w_{15}, v_k) = 84 \times 20 = 1{,}680$ *bytes.*

$Size(w_1, v_i)$:

$|f_{1,d}\,[PATIENT\_NO]f_{12,i}\,|$

$$= \left\lceil \frac{5075 \times 84}{126} \right\rceil = 3{,}383 \textit{ tuples.}$$

$|f_{1,b}\,[ORDER]| \times |f_{12,i}\,[NAME]| = 5075 \times 84 > 3{,}383,$

therefore $|(f_{1,d}\,[PATIENT\_NO]f_{12,i}\,)[ORDER,NAME]|$

$= 3{,}383 \times (4 + 15) = 64{,}277$ *bytes.*

$Size(w_1, v_k)$:

$|(f_{1,d}\,[PATIENT\_NO]f_{12,i}\,)[NAME]f_{15,k}\,|$

$$= \left\{ \frac{\left\lceil \frac{5075 \times 84}{126} \right\rceil \times 12}{126} \right\} = 322 \textit{ tuples.}$$

$|f_{1,b}\,[ORDER]| \times |f_{15,k}\,[WARD]| = 5{,}075 \times 1 = 5075 > 322,$

therefore $|((f_{1,d}\,[PATIENT\_NO]f_{12,i}\,)[NAME]f_{15,k}\,)[ORDER, WARD]|$

$= 322 \times (4 + 5) = 2{,}898$ *bytes.*

$Size(w_{12}, v_k)$:

$$|f_{12,i}\,[NAME]f_{15,k}\,| = \left\lceil \frac{84 \times 12}{126} \right\rceil = 8 \textit{ tuples.}$$

$|f_{12,i}[NAME]| \times |f_{15,k}[WARD]| = 84 \times 1 = 84 > 8,$

therefore $Size(w_{12}, v_k) = 8 \times (4 + 5) = 72$ *bytes.*

(10) Metagraph (**Y**,**L**) is shown in Figure 4.13 with the weights of all the arcs. (Note that the weight of an arc in **L** is kept to be the size of the FD that **L** stands for, rather than a large number minus that size).

The application of Algorithm *OutTree* on (**Y**,**L**) is straightforward, and the result is shown in thick lines in Figure 4.13. Note that arc L_{s1} representing FD *ORDER* → *PATIENT_NO* must be included in any feasible solution because of the lossless join requirement.

The envelope for $Q_{t,5}$ is:

$ACC_{t,5} = \{f_{1,d}, f_{12,i}, f_{12,k}\}.$

(11-12) All the feasible partitions of $ACC_{t,5}$ are shown in Table 4.3. $PAR_{t,5} = \{1, 2, 3, 4, 5\}$ and, for example, $q = 2 \in CLU_t{}^s{}_5$ corresponds to relation scheme $<\{PATIENT\_NO, NAME, WARD\}$, $\{PATIENT\_NO \rightarrow NAME,\ NAME \rightarrow WARD\}>$.

(14) For each partition of $ACC_{t,5}$ adistributed query processing algorithm such as [CHUN 83]'s is run. In Appendix F, input file $query_t{}^2{}_5$, the input file for "Algorithm H" [CHUN 83] corresponding to partition $p = 2$ of $ACC_{t,5}$ is shown.

(15) "Algorithm H" [CHUN 83] is then run with respect to relation schemes $<ORDER, PATIENT\_NO, NAME, >$ and $<PATIENT\_NO, WARD, >$ since Table 4.3 indicates that these are the clusters to consider for partition $p = 2$.

For this example, the output is:

$A_{t,5}^{2,1} = 88.936$ *kilobits* $\times\ \nu_{t,5}$.

$A_{t,5}^{2,2} = 0.864$ *kilobits* $\times\ \nu_{t,5}$.

p	$q \in CLU_{t,5}^{p}$	$PAR_{t,5}$
1	1	<ORDER,PATIENT_NO,NAME,WARD>
2	1	<ORDER,PATIENT_NO,NAME>
	2	<PATIENT_NO,WARD>
3	1	<ORDER,PATIENT_NO>
	2	<PATIENT_NO,NAME,WARD>
4	1	<ORDER,PATIENT_NO,WARD>
	2	<PATIENT_NO,NAME>
5	1	<ORDER,PATIENT_NO>
	2	<ORDER,NAME>
	3	<PATIENT_NO,WARD>

Table 4.3. Possible partitions of the referenced attributes for query $Q_{t,5}$.

$\bar{A}_{t,\ 5}^{2,\ 1,2} = 0.464$ *kilobits* $\times\ \nu_{t,5}$.

$\bar{A}_{t,\ 5}^{2,\ 1,2} = A_{t,\ 5}^{2,\ 1,2} = A_{t,\ 5}^{2,\ 2,1} = 0.$

For example, if site 3 is the origine of the query, then $t = 3$, and from Table C.3 $\nu_{3,5} = 3$ $(1/minutes)$. Therefore:

$A_{3,\ 5}^{2,\ 1} = 266.808$ *kilobits/minutes.*

$A_{3,\ 5}^{2,\ 2} = 2.592$ *kilobits/minutes.*

$A_{3,\ 5}^{2,\ 2,1} = 1.392$ *kilobits/minutes.*

Let steps (3) through (12) be repeated for query $Q_{t,7}$.

(3) Query $Q_{t,7}$ is expressed as:

$Q_{t,7}$: FIND (*RESULT, ORDER, TEST*)

$C_{t,7}$: WHERE *NAME* = ' *Rowlands*'

AND *DATE* = ' 06-20-83' .

The target list of $Q_{t,7}$ is {*RESULT, TEST, ORDER, NAME, DATE*}. The set of target nodes is $T = \{v_a, v_g, v_c, v_i, v_e\}$. By applying Algorithm *GetSource*, *Source* is found to be $\{w_1\}$.

(4) $w_s = w_1$ since this is the only source node that leads to all the nodes in T.

(5) *GetLink* produces $L(w_s, T) = L(w_1, \{v_a, v_c, v_e, v_g, v_i\})$. $L(w_s, T)$ for query $Q_{t,7}$ is shown in Figure 4.14.

(6) The link for $Q_{t,7}$ depicts the following FDs:

$f_{1,c}$: *ORDER* → *TEST*

$f_{1,d}$: *ORDER* → *PATIENT_NO*

$f_{1,e}$: *ORDER* → *DATE*

$f_{11,g}$: *TEST,PATIENT_NO,DATE* → *RESULT*

$f_{12,i}$: *PATIENT_NO* → *NAME.*

When those FDs are restricted by the qualification clause $C_{t,7}$ (*NAME* = ' *Rowlands*' AND *DATE* = ' 06-20-83') the statistics of Ω are modified by *LocalProc* as shown in Tables 4.4 and 4.5. Also $|DOM(DATE)| = 1$, $|DOM(NAME)| = 1$, and $|DOM(TEST,PAT\_NO,DATE)| = 150$.

(7) This step is not relevant.

FD	$\lvert f \rvert$ *tuples*	$\bar{f}$ *bytes*
$f_{1,c}$	5075	14
$f_{1,d}$	5075	8
$f_{1,e}$	318	12
$f_{11,g}$	324	44
$f_{12,i}$	1	19

Table4.4. Cardinalities and widths of the FDs belonging to the envelope of query $Q_{t,7}$.

$\lvert f[A] \rvert$ *tuples* where A is ...						
FD	ORDER	TEST	PAT_NO	DATE	RESULT	NAME
$f_{1,c}$	5075	76				
$f_{1,d}$	5075		84			
$f_{1,e}$	5075			632		
$f_{11,g}$		70	15	1	5432	
$f_{12,i}$			1			1

Table 4.5 Cardinalities of the FDs, in the envelope of query $Q_{t,7}$, projected on each referenced attribute.

(8) The metagraph of Figure 4.15 is obtained.

The arcs are defined as followed:

$L_{s1} \overset{\Delta}{=} [<v_a, w_1> ; <\{v_c, v_d, v_e\}, w_{11}>]$

$\equiv \{v_a, (v_a, w_1), w_1, (w_1, v_c), (w_1, v_d), (w_1, v_e), v_c, v_d, v_e\}$

$L_{s2} \overset{\Delta}{=} [<v_a, w_1> ; <v_d, w_{12}>]$

$\equiv \{v_c, (v_a, w_1), w_1, (w_1, v_d), v_d\}$

$L_{12} \overset{\Delta}{=} [<\{v_c, v_d, v_e\}, w_{11}> ; <v_d, w_{12}>]$

$\equiv \{v_d\}$

$L_{13} \overset{\Delta}{=} [<\{v_c, v_d, v_e\}> ; <v_g, \emptyset>]$

$\equiv \{v_c, v_d, v_e, (v_c, w_{11}), (v_d, w_{11}), (v_e, w_{11}), w_{11}, (w_{11}, v_g), v_g\}$

$L_{24} \overset{\Delta}{=} [<v_d, w_{12}> ; <v_i, \emptyset>]$

$\equiv \{v_d, (v_d, w_{12}), (w_{12}, v_i), v_i\}$

$L_{s3} \overset{\Delta}{=} [<v_a, w_1> ; <v_g, \emptyset>]$

$\equiv \{v_a, (v_a, w_1), w_1, (w_1, v_c), (w_1, v_d), (w_1, v_e), v_c, v_d, v_e,$

$(v_c, w_{11}), (v_d, w_{11}), (v_e, w_{11}), w_{11}, (w_{11}, v_g), v_g\}$

$L_{s4} \overset{\Delta}{=} [<v_a, w_1> ; <v_i, \emptyset>]$

$\equiv \{v_a, (v_a, w_1), w_1, (w_1, v_d), v_d, (v_d, w_{12}), w_{12}, (w_{12}, v_i), v_i\}$.

Further, the arcs in **L** are associated with implicit or explicit FDs as follows:

$L_{s1} \sim f_{1,c} : ORDER \rightarrow TEST$

$f_{1,d} : ORDER \rightarrow PATIENT\_NO$

$f_{1,e} : ORDER \rightarrow DATE$

$L_{s2} \sim f_{1,d} : ORDER \rightarrow PATIENT\_NO$

L_{12} ˜ $f_{11,d}$: *TEST,PATIENT_NO,DATE* → *PATIENT_NO*

L_{13} ˜ $f_{11,g}$: *TEST,PATIENT_NO,DATE* → *RESULT*

L_{24} ˜ $f_{12,i}$: *PATIENT_NO* → *NAME*

L_{s3} ˜ $f_{1,g}$: *ORDER* → *RESULT*

L_{s4} ˜ $f_{1,i}$: *ORDER* → *NAME.*

(9) **The sizes of the FDs in (8) are calculated by *SizeEstimate* as follows (the scripts $(t,k) = (t,7)$ are omitted in the notations):**

$Size(w_1,\{v_c, v_d, v_e\})$:

$$|f_{1,c}[ORDER]f_{1,d}| = \frac{5075 \times 5075}{5075} = 5075 \textit{ tuples.}$$

$$|(f_{1,c}[ORDER]f_{1,d})[ORDER]f_{1,e}| = \frac{5075 \times 318}{5075} = 318 \textit{ tuples.}$$

$|((f_{1,c}[ORDER]f_{1,d})[ORDER]f_{1,e})[ORDER,TEST,PAT\_NO,DATE]|$

$= 318 \times (4 + 10 + 4 + 8) = 8{,}268$ *bytes.*

$Size(w_1,v_d) = 5075 \times 8 = 40{,}600$ *bytes.*

$Size(w_{11},v_d) = 0$ *bytes.*

$Size(w_{11},v_g) = 324 \times 44 = 14{,}256$ *bytes.*

$Size(wub12,v_i) = 1 \times 19 = 19$ *bytes.*

$Size(w_1,v_g)$:

$|((f_{1,c}[ORDER]f_{1,d})[ORDER]f_{1,e})[TEST,PAT\_NO,DATE]f_{11,g}|$

$$= \frac{318 \times 324}{150} = 687 \textit{ tuples.}$$

$|f_{1,c}[TEST]|$ '$\times$ $|f_{1,d}[PAT\_NO]| \times |f_{1,e}[DATE]|$

$= 76 \times 84 \times 632 > 318$, and $318 \times 5432 > 687$ therefore:

$Size(w_1, v_g) = 687 \times (10 + 4 + 8 + 22) = 30{,}228$ *bytes.*

$Size(w_1, v_i)$:

$|f_{1,d}[PATIENT\_NO]f_{12,i}| = \frac{5075 \times 1}{126} = 40$ *tuples.*

$|f_{1,d}[ORDER]| \times |f_{12,i}[NAME]| = 5075 \times 1 > 40$

therefore $Size(w_1, v_i) = 40 \times (4 + 15) = 760$ *bytes.*

(10) **Metagraph (Y,L) is shown in Figure 4.16 with the weights (sizes) of all the arcs.**

The envelope for $Q_{t,7}$ is:

$ACC_{t,7} = \{f_{1,c}, f_{1,d}, f_{1,e}, f_{11,g}, f_{11,d}, f_{12,i}\}$.

Strictly speaking, $f_{1,c}$, $f_{1,d}$ and $f_{1,e}$ are three distinct FDS, however, since they have the same key they may be combined into their natural join over attribute *ORDER*. $f_{1,c,d,e}$ denotes the join of $f_{1,c}$, $f_{1,d}$ and $f_{1,e}$ over *ORDER*. Note also that $f_{11,d}$ is a "dummy" FD, and therefore the envelope of $Q_{t,7}$ really is:

$ACC_{t,7} = \{f_{1,c,d,e}, f_{11,g}, f_{12,i}\}$.

(11-12) **All the feasible partitions of $ACC_{t,7}$ are shown in Table 4.6. For example, $q = 3 \in CLU_{t,7}^{5}$ corresponds to relation scheme $<\{PAT\_NO,NAME\},\{PAT\_NO \rightarrow NAME\}>$.**

An update example is now worked out. Let the update order be:

$U_{s,2}$: **ADD** $<PATIENT\_NO,NAME> = <7785,Smith,45>$.

This is identical to update 2 in Appendix C without reference to attribute *AGE*.

Suppose the following information has previously been added:

$<ORDER,TEST,RESULT,DATE> = <18,Glucose,70,06\text{-}21\text{-}83>$, **and**

$<NAME,WARD> = <Smith,Outpatient>$.

p	$q \in CLU^p_{t,7}$	$PAR_{t,7}$
1	1	<ORDER,TEST,PAT_NO,DATE,RESULT,NAME>
2	1	<ORDER,TEST,PAT_NO,DATE,RESULT>
	2	<PAT_NO,NAME>
3	1	<ORDER,TEST,PAT_NO,DATE>
	2	<TEST,PAT_NO,DATE,RESULT,NAME>
4	1	<ORDER,TEST,PAT_NO,DATE,NAME>
	2	<TEST,PAT_NO,DATE,RESULT>
	3	<PAT_NO,NAME>

Table 4.6 Possible partitions of the attributes referenced by query $Q_{t,7}$.

Assume that the F-graph of Figure 4.9 depicts the final FDs to be distributed, i.e. the FDs of H the set of all the query envelopes.

(16) $DEL_{s,2} = \{(w_{12}, v_i)\}$, since (w_{12}, v_i) represents FD $PATIENT\_NO \rightarrow NAME$ which itself features the attributes referenced by $U_{s,2}$.

(17) Initially, $ALTER = \{(w_{12}, v_i)\}$.

$\bar{L}_i^+ = \{v_a, (v_a, w_1), w_1, (w_1, v_d), v_d, (v_d, w_{12}),$

$w_{12}, (w_{12}, v_i), v_i, (v_i, w_{15}), w_{15}, (w_{15}, v_k), v_k\}$.

$\bar{L}_i^- = \{v_a, (v_a, w_1), w_1, (w_1, v_b), v_b, (v_b, w_8), w_8,$

$(w_8, v_f), v_f, (v_f, w_{10}), w_{10}, (w_{10}, v_k), v_k\}$.

Note that both $\bar{L}_i^+$ and $\bar{L}_i^-$ are proper links from w_1 to v_k.

$\{(w_{12}, v_i)\} \subset \bar{L}_i^+.$

$$d_{t_i} = \Big\{ I_{<ORDER,PAT\_NO,NAME,WARD,>}$$

$$\Big[PAT\_NO, NAME, WARD,\ \ t[PAT\_NO, NAME] = d_{t_i} = <7785, Smith> \Big] \Big\}$$

$[WARD] = Outpatient.$

$$r_{s_i} = \Big\{ I_{<ORDER,PAT\_NO,NAME,WARD,>}$$

$$\Big[ORDER, PAT\_NO,\ \ t[PAT\_NO] = 7785 \Big] \Big\} [ORDER]$$

$= \{18\}.$

$k = 0,\ \ d_0 = Outpatient,\ \ r[0] = \{Outpatient\}.$

$X_0^- = WARD,\ \ X_0 = PHYSICIAN.$

$$r[1] = \Big\{ I_{<ORDER,SERVICE,PHYSICIAN,WARD>}$$

$$\Big[PHYSICIAN, WARD,\ \ WARD = Outpatient \Big] \Big\} [PHYSICIAN]$$

$= \{Dr.\ Akers\}.$

$$r[2] = \Big\{ I_{<ORDER,SERVICE,PHYSICIAN,WARD,>}$$

$$\Big[SERVICE, PHYSICIAN,\ \ PHYSICIAN = Dr.\ Akers \Big] \Big\} [SERVICE]$$

$= \{S\text{-}1,\ S\text{-}3\}.$

$$r[3] = \Big\{ I_{<ORDER,SERVICE,PHYSICIAN,WARD>}$$

$$\Big[ORDER, SERVICE,\ \ SERVICE \in \{S\text{-}1,\ S\text{-}3\} \Big] \Big\} [ORDER]$$

$= \{05,\ 07,\ 10,\ 15\}.$

$r_{s_i} = 18 \notin \{05, 07, 10, 15\}$,

therefore, there is no antecedent for some cutset in $\bar{L}_i^-$.

$s[3] = \{18\}$, $X_3 = ORDER$, $X_{\bar{3}} = SERVICE$.

$$s[2] = \left\{ I_{<ORDER,SERVICE,PHYSICIAN,WARD,>} \left[ORDER,SERVICE,\ ORDER = 18 \right] \right\}[SERVICE]$$

$= \emptyset$.

$q[2] = \{S\text{-}1, S\text{-}3\}$, therefore $ORDER = 18$ should be assigned $SERVICE = S\text{-}1$ or $SERVICE = S\text{-}3$. Suppose Mr Smith is in service S-1, then the tuple $<ORDER,SERVICE> = <18,S\text{-}1>$ must be added to the database.

Thus $ALTER := ALTER \bigcup \{(w_1,v_b)\}$. Hence, for this particular update, the cascade is indicated by:

$ALTER = \{(w_{12},v_i), (w_1,v_b)\}$.

Any update of $PATIENT\_NO \rightarrow NAME$ is to be followed by an update of $ORDER \rightarrow SERVICE$.

$WRT_{s,2} = \{f_{12,i}: PATIENT\_NO \rightarrow NAME, f_{1,b}: ORDER \rightarrow SERVICE\}$.

If the user site is site $s = 4$. From Table C.3 $\nu_{4,2} = 2$ $(1/minutes)$. Therefore:

$$B_{4,\ 2}^{12,\ i} = (4 + 15\ bytes\ +\ 2\ bytes) \times 2\ \frac{1}{minutes},$$

$$= 42\ \frac{bytes}{minutes}.$$

Similarly, $B_{4,\ 2}^{1,\ b} = (4 + 5 + 2) \times 2 = 22\ \frac{bytes}{minutes}$.

(It was assumed that $\mu = 2$ $bytes$. The widths of the attributes are provided by Table C.6).

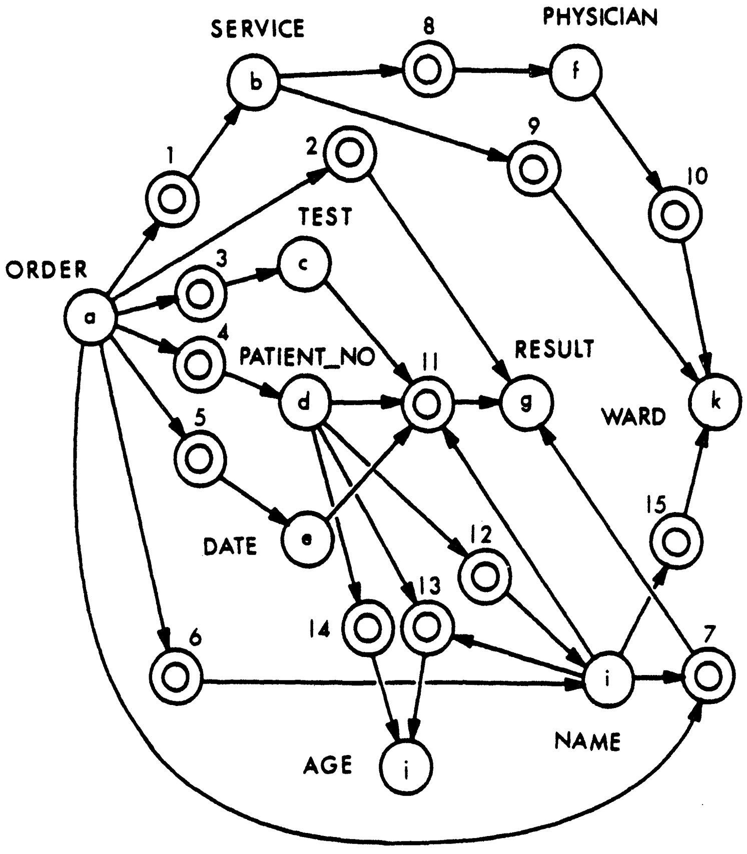


Figure 4.8. An F-graph depiction of the Medical Record Database

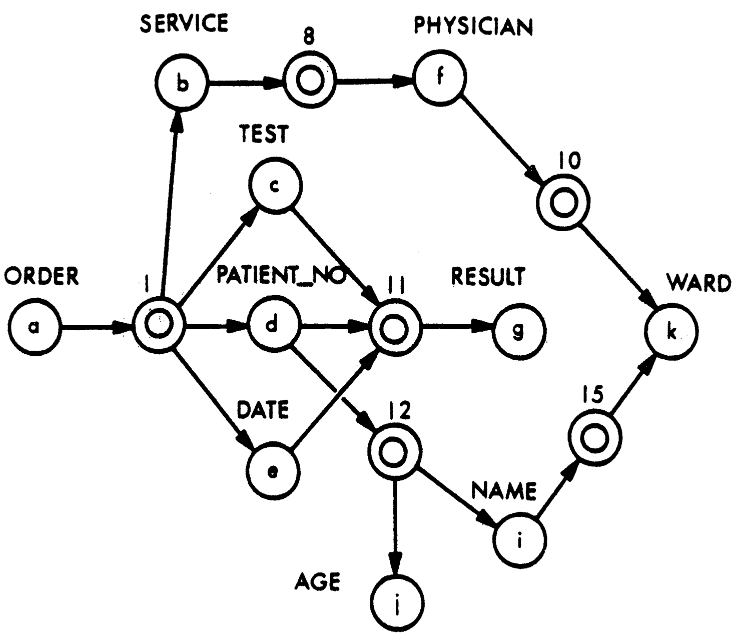


Figure 4.9. A minimum F-graph for the MRDB

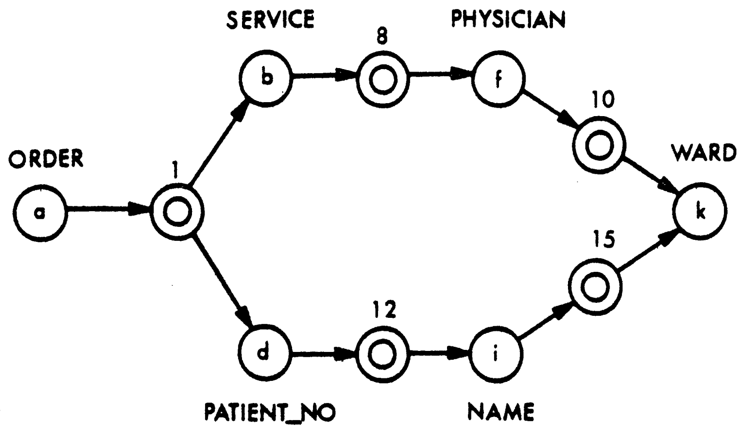


Figure 4.10. The link of query $Q_{t,5}$

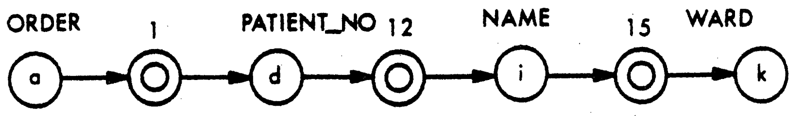
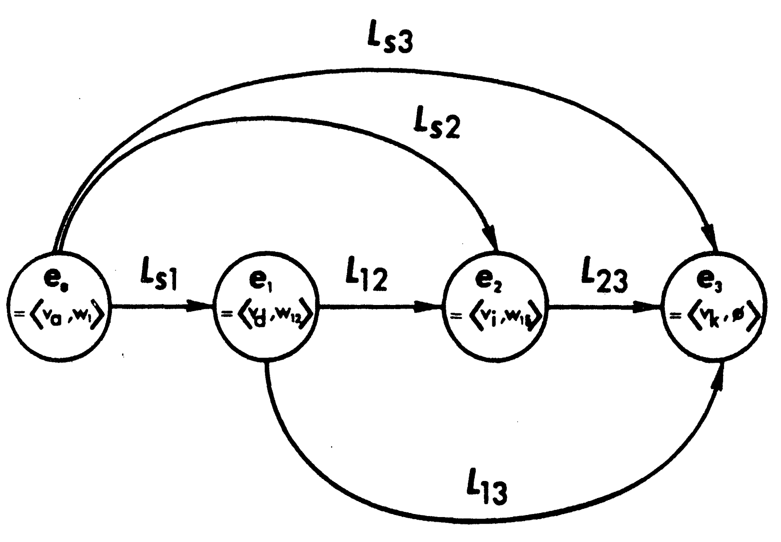


Figure 4.11. A proper link for query $Q_{t,5}$



L_{s3}
L_{s2}
$e_s = \langle v_a, w_1 \rangle$
L_{s1}
$e_1 = \langle v_d, w_{12} \rangle$
L_{12}
$e_2 = \langle v_i, w_{18} \rangle$
L_{23}
$e_3 = \langle v_k, \emptyset \rangle$
L_{13}

Figure 4.12. The metagraph for query $Q_{t,5}$

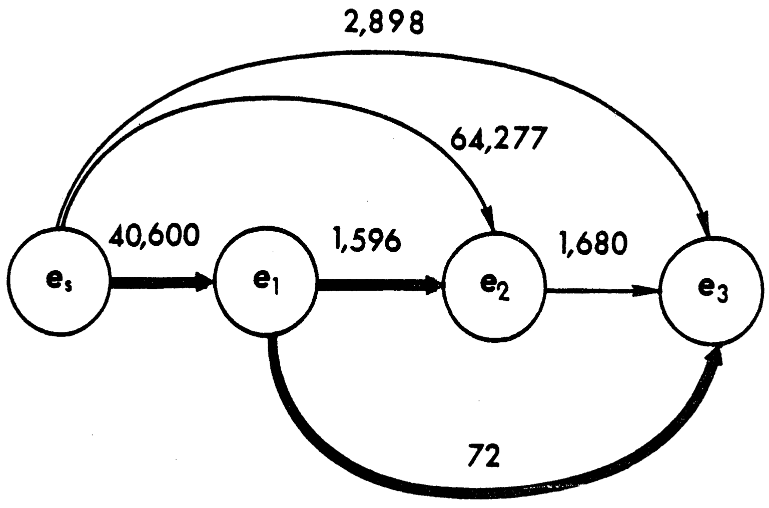


Figure 4.13. The weighted metagraph for query $Q_{t,5}$

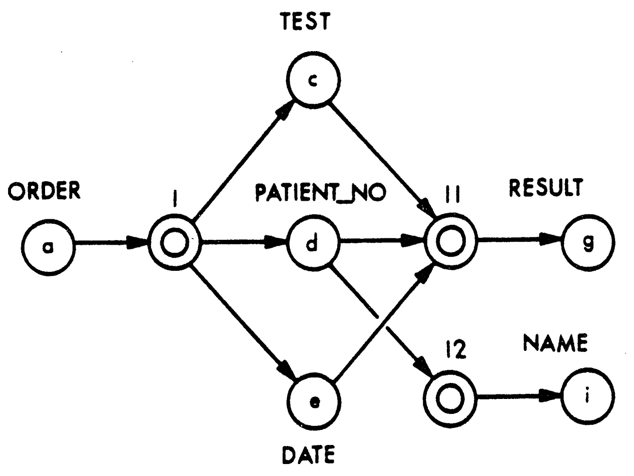


Figure 4.14. The link for query $Q_{t,7}$

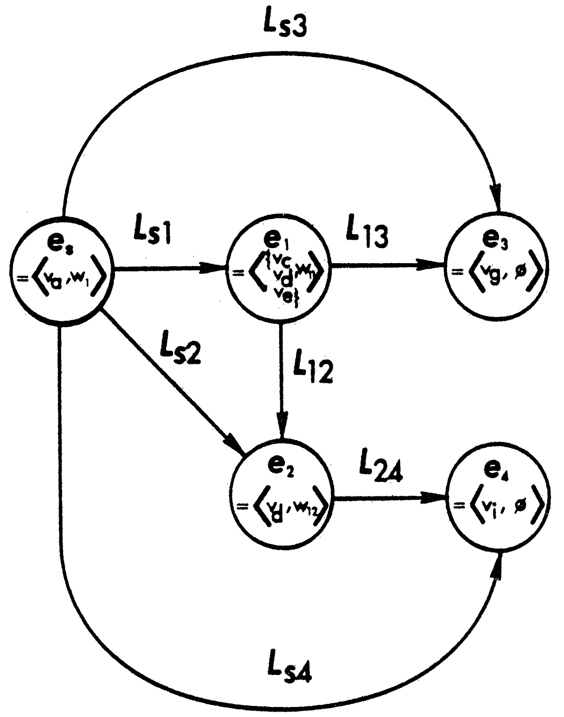

L_{s3}
$e_s = \langle v_a, w_1 \rangle$
L_{s1}
$e_1 = \langle \{v_c, v_d, v_e\}, w_{11} \rangle$
L_{13}
$e_3 = \langle v_g, \emptyset \rangle$
L_{s2}
L_{12}
$e_2 = \langle v_f, w_{12} \rangle$
L_{24}
$e_4 = \langle v_i, \emptyset \rangle$
L_{s4}

Figure 4.15. The metagraph for query $Q_{t,7}$

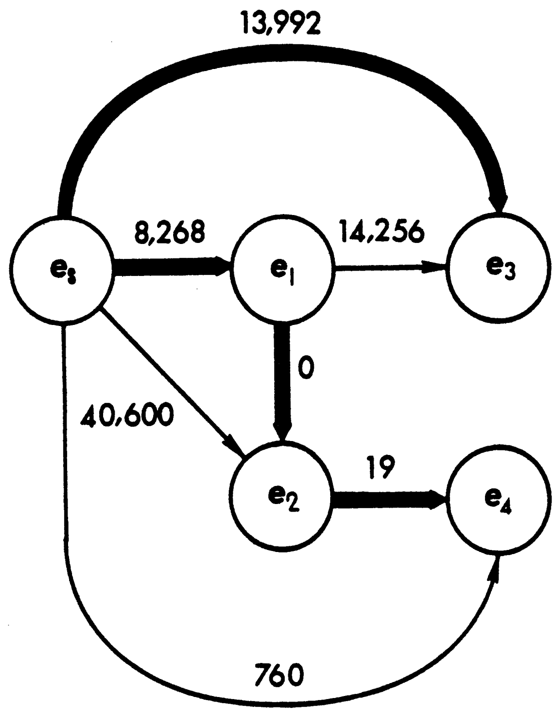


Figure 4.16. The weigthed metagraph for query $Q_{t,7}$

CHAPTER 5

A MODEL FOR DATABASE DISTRIBUTION

The cross-referencing data units (CRDUs) contained in all query envelopes are to be distributed (with possible duplication) over the nodes of a computer network. In this chapter, a CRDU-distribution model is proposed. The objective is to minimize the communication cost. It is assumed that all the transactions will be processed according to distributed query processing strategies that feature the cross-referencing of relations as explained in Chapter 2. For a single distributed query processing session, some flow is produced by file-to-file communications, and is associated with the reducing operations. Some flow is produced by file-to-user communications, and is associated with the assembly operations. It is assumed that the assembly node, the node where the answer of a query is assembled, is the user node itself. The sizes of the initial CRDUs are estimated as demonstrated in Chapter 4. For each query Q_{tk}, an envelope can be determined as shown in Chapter 4, and the set of CRDUs mapped to the FDs of the query envelope is denoted ACC_{tk}. The sizes of the CRDUs contained in ACC_{tk}, after local processing, may be obtained by applying the "selection" relational operator implied by the user-provided qualification clauses contained in C_{tk}. Furthermore, it is assumed that an appropriate distributed processing algorithm can be applied on each feasible partition of the set of CRDUs ACC_{tk} to determine the data-volume flow rates which will serve as inputs for this part of the distributed database design. Each feasible partition p of a set of CRDUs ACC_{tk} consists in a set CLU^p_{tk} of clusters. Each cluster can be visualized as one individual relation. Therefore, a query Q_{tk} can be solved as a distributed query over the distinct relations of $CLU_{t k}^{p}$. After distribution, a synthesis algorithm such as [KAMB 78] can be used to combine FDs in third normal form relational tables. Assumptions regarding the model are summarized in Appendix C.

The next section contains a notation list with concise definitions. In Section 5.2 a so-called "naive" distribution model is developped. This naive model is to enlighten the reader as to the difficulties of distributing CRDUs. Typically, the clustering-effect discussed in Chapter 2 is at the source of the problems that arise when the modeling of data-volume flow rates is attempted.

5.1. Notations

U	Universe of attributes.
H	Set of CRDUs to be distributed.
N	Set of nodes, or sites, in a computer network.
A	Set of arcs, or network links, in a computer network.
c_{ij}	Communication cost for arc $(i,j) \in A$.
$\bar{c}_{ij}$	Communication cost corresponding to the shortest path from i to j in network (N,A).
K_{ij}	Flow capacity of link (i,j) in network (N,A).
t	Usually stands for a single user-node, $t \in N$. At the same time t denotes a *sink* node for some flow.
Q_t	Set of queries originated at user-node t. $\overset{\Delta}{=} \{Q_{tk} : 1 \leq k \leq K_t\}$ for some K_t.
Q_{tk}	k^{th} query originated at user-node t.

S_{tk}	Set of nodes in N that user-node t needs access to before answering Q_{tk}. Only used in the naive distribution model.
$\overline{S}_{tk}$	Subset of S_{tk} such that there exists a reducing flow from $s \in S_{tk}$ to all the nodes of $\overline{S}_{tk}$ and possibly to the user-node t. Only used in the naive distribution model.
s	Usually denotes a *source* node for some flow.
U_{sk}	k^{th} update originated at user-node s.
ACC_{tk}	Envelope of query Q_{tk}, i.e. set of CRDUs needed to process Q_{tk}.
WRT_{sk}	Cascade of update U_{sk}, i.e. set of CRDUs to be updated by U_{sk}.
PAR_{tk}	Set of feasible partitions of an envelope ACC_{tk}.
$CLU_t^p{}_k$	One feasible partition of ACC_{tk} corresponding to a partitioning label $p \in PAR_{tk}$.
p	Denotes a partition label, i.e. one element of some PAR_{tk}.
q, r	Denote clusters in some set of clusters $CLU_t^p{}_k$. Each cluster q or r stands for some relation obtained by joining CRDUs of ACC_{tk}.
$A_t^p{}_k^q$	Data-volume flow rate corresponding to the shipment for assembly of relation q to a user-node (sink-node) t after all the local processing and all the reducing operations have been completed. q is some relation, or cluster, of $CLU_t^p{}_k$ which is itself the p^{th} feasible partition of the envelope ACC_{tk} ($p \in PAR_{tk}$) of some query Q_{tk} originated at user-node t.
$A_t^p{}_k^{qr}$	Data-volume flow rate corresponding to the shipment of all the reducing data sent from relation q to relation r of $CLU_t^p{}_k$ in the context of some critical file-to-file cross-referencing strategy.
$\overline{A}_t^p{}_k^{qr}$	As above, but strictly for the noncritical reducing operation from relation q to relation r.

N_h	Number of relations q that contain CRDU $h \in H$ over all clusters, partitions, and queries.
f_g	Denotes a CRDU held by a source node with respect to some flow. Only used in the naive distribution model.
$S(f)$	Attribute scheme of FD f.
$R_{tk}(f_g)$	Covering of some envelope ACC_{tk} such that $f_g \in ACC_{tk}$ sends reducing data (semi-joining columns of relations) to it. $\overset{\Delta}{=} \{h : h \in ACC_{tk} - \{f_g\}$ & *CRDU f_g sends reducing data to CRDU h* $\}$. Only used in the naive distribution model.
$A_{t}^{g}{}_{k}$	Data-volume flow rate corresponding to the shipment of CRDU $f_g \in ACC_{tk}$ from some source-node s to node t after all the local processing and all the reducing operations have been performed. Contributes to the assembly flow for query Q_{tk}. Only used in the naive distribution model.
$A_{t}^{g}{}_{k}^{h}$	Data-volume flow rate corresponding to the shipment of some reducing data from CRDU $f_g \in ACC_{tk}$ to CRDU $f_h \in ACC_{tk}$. Contributes to the reducing flow for query Q_{tk}. Only used in the naive distribution model.
$B_{s}^{h}{}_{k}$	Data-and-message-volume flow rate corresponding to some updating information sent from user-node s to CRDU $f_h \in WRT_{sk}$ for some update U_{sk}.
$a_{t}^{g}{}_{ks}$	Flow rate which corresponds to the shipment, for assembly, of a processed copy of CRDU $f_g \in ACC_{tk}$ located at source-node $s \in S_{tk}$, to user-node t, the sink-node. Only used in the naive distribution model.
$i_{t}^{g}{}_{ks}$	Flow rate which corresponds to the shipment of reducing data from CRDU $f_g \in ACC_{tk}$ located at $s \in S_{tk}$, to all the nodes in $\bar{S}_{tk}$ holding CRDUs involved in cross-referencing with f_g. Only used in the naive distribution model.

$x_{t\,ks}^{g}$	Flow rate corresponding to the total reducing and assembly flow rates corresponding to the shipment of some data of CRDU $f_g \in ACC_{tk}$ located at $s \in S_{tk}$. Only used in the naive distribution model.
ASM_{si}	Assembly flow-rate value at node i, due to the total assembly flow with source node s.
RDC_{si}	Reducing flow-rate value at node i, due to the total file-to-file flow with source node s.
UPD_{si}	Update flow-rate value at node i, due to the total update flow with source node s.
x_s	Total flow-rate with over all the flows with source node s.
Y_i^h	$\triangleq \begin{cases} 1 & \text{if } CRDU\ h \in H \text{ is located at node } i \in N \\ 0 & \text{otherwise.} \end{cases}$
Y_h	$\triangleq \sum_{j=1}^{\vert N\vert} Y_j^h$
$Z_{t\,ki}^{h}$	$\triangleq \begin{cases} 1 & \text{if node } i \text{ holds a copy of } h \in H \text{ for } Q_{tk}, \\ 0 & \text{otherwise.} \end{cases}$ Only used in the naive distribution model.
$X_{t\,k}^{p}$	$\triangleq \begin{cases} 1 & \text{if partition } p \in PAR_{tk} \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases}$
$X_{t\,ki}^{pq}$	$\triangleq \begin{cases} 1 & \text{if node } i \text{ holds cluster } q \text{ for query } Q_{tk} \\ & \text{and partition } p \in PAR_{tk} \\ 0 & \text{otherwise.} \end{cases}$
w_{ti}	$\triangleq \begin{cases} 1 & \text{if } i \equiv t, \\ 0 & \text{otherwise.} \end{cases}$
$x(i,N)$	$\triangleq \sum_{j=1}^{\vert N\vert} x(i,j)$ where $x(i,j)$ denotes any type of flow rate from node i to node j.

The next section deals with the "naive" distribution model.

5.2. "Naive" distribution modeling

The following "naive" modeling approach is to stress the difficulty of capturing the clustering effect discussed earlier in Chapter 2. Each user-node t is associated with a set of queries $Q_t \overset{\Delta}{=} \{Q_{tk} : 1 \leq k \leq K_t\}$ for some K_t. For each query Q_{tk} corresponds an envelope $ACC_{tk} \subseteq H$. Let $S_{tk} \subseteq N$ denote the unknown set of nodes in N that user-node t needs to communicate with in the context of a distributed query processing session aiming at answering Q_{tk}. Let $s \in S_{tk}$ be the node that holds a copy of CRDU $f_g \in ACC_{tk}$. Further, let $\overline{S}_{tk}(s) \subseteq S_{tk}$ be such that for any node $\overline{s}$ in $\overline{S}_{tk}(s)$ there exists some file-to-file communication from s to $\overline{s}$. Two types of flows resulting from the access of CRDU f_g may be considered:

(1) An assembly flow, denoted $a_t{}^g{}_{ks}$, taking source at node s and terminating at node t, such that for any $s \in S_{tk}$, where s carries a copy of CRDU f_g, the following flow equations hold:

$$- a_t{}^g{}_{ks}(j,N) + a_t{}^g{}_{ks}(N,j)$$

$$= - A_t{}^g{}_k \qquad \text{for } j = s,$$

$$= \ 0 \qquad \text{for any } j \notin \{s\} \cup \{t\},$$

$$= \ A_t{}^g{}_k \qquad \text{for } j = t.$$

It was assumed that $s \neq t$.

(2) A reducing flow, denoted $i_t{}^g{}_{ks}$, taking source at node s and terminating at nodes of $\overline{S}_{tk}(s)$. For all $s \in S_{tk}$ such that s carries CRDU f_g and such that the set of nodes with which s communicates for cross-referencing is $\overline{S}_{tk}(s)$, the following flow equations hold:

$$- i_t{}^g{}_{ks}(j,N) + i_t{}^g{}_{ks}(N,j)$$

$$= - \sum_{f_h \in R_{tk}(f_g)} A_t{}^g{}_k^h \qquad \text{for } j = s,$$

$= \; 0$ for any $j \notin \{s\} \bigcup \bar{S}_{tk}(s)$,

$= \; A_{f\,k}^{g\,h}$ for any $j \in \bar{S}_{tk}(s)$

and j carries a copy of $f_h \in ACC_{tk}$

This is so assuming that all the CRDUs in ACC_{tk} are held in different nodes, i.e. that $|\bar{S}_{tk}(s)| = |ACC_{tk}| - 1$.

The assembly and reducing flows may be summed up. It is meaningful to do so as long as only the flows with identical sources are dealt with. Thus for a distributed query processing session aimed at answering a query Q_{tk}, the total flow is such that:

$$- x_{f\,ks}^{g}(i,N) + x_{f\,ks}^{g}$$

$$= \left[- A_{f\,k}^{g} - \sum_{f_h \in R_{tk}(f_g)} A_{f\,k}^{g\,h} \right] Z_{f\,ks}^{g} \, w_{si}$$

$$+ A_{f\,k}^{g} \, w_{ti}$$

$$+ \sum_{f_h \in R_{tk}(f_g)} A_{f\,k}^{g\,h} \, Z_{f\,ki}^{h} \, Z_{f\,ks}^{g}, \quad \text{for any } i \in N.$$

Given a source-node s, all such flows may be added up over all user-nodes t, all queries Q_{tk}, and all CRDUs $f_g \in ACC_{tk}$. However, the development of the naive distribution model is interrupted at this point to point out a difficulty that brings up the need for a more complete model.

Example: Consider Figure 5.1. Suppose that node s which holds a copy of f_g needs to seek access to h_1 and h_2 to perform semi-joins $f_g[A>h_1$ and $f_g[A>h_2$, both over attribute A. Assume that both CRDUs h_1 and h_2 are placed at node j. Let $A_{f\,k}^{g\,h1}$ and $A_{f\,k}^{g\,h2}$ be the data-volumes corresponding respectively to semi-joins $f_g[A>h_1$ and $f_g[A>h_2$. It is clear that the shipment of $A_{f\,k}^{g\,h1} + A_{f\,k}^{g\,h2}$ units of data-volume from node s to node j would be quite inefficient. Indeed, this strategy would imply that a column of A values is sent *twice* from s to j. This problem may be summarized thus: *The placement, at the same site, of two or more CRDUs which contain common informations result in drastic nonlinearity in data-volume flows.* Earlier in Chapter 2, this problem was identified as the *clustering effect*. It is therefore quite inappropriate to deal with individual CRDUs such as f_g in a flow model. Rather, clusters of CRDUs must be considered to allow the input of meaningful data-volumes.

•

In the next section, the effect of clustering is taken into consideration as well as the quality of semi-joins operations, i.e. whether they are critical or not.

5.3. PART 2: A model for the distribution of CRDUs

The CRDU-distribution problem is captured in the form of a mathematical optimization model. The assumptions on which the final model for the combined distribution and materialization of CRDUs is based are as follows. (1) The underlying communication network of the distributed system has a fixed topology. (2) Appropriate distributed query processing sequences can be determined for all possible clustering configurations of CRDUs. An optimization algorithm, which will be part of the query processing subsystem during the operational phase of the distributed database, is readily available [CHUN 83] (see Sections 4.6 and 4.7). (3) The distributed query processing strategies are based on the principle of static materialization. (4) Directories, programs and materialization look-up tables are available at each site. (5) Effects of concurrency control mechanisms are not considered.

The following are assumed to be given:

(N,A)	The underlying directed graph of a computer network.
$c_{ij} = c_u$	Communication cost for arc e_u = (i,j) representing a communication link.
$\overline{c}_{ij}$	Communication cost corresponding to the shortest path from i to j.
$K_{ij} = K_u$	Flow capacity of communication link e_u = (i,j).
H	Set of CRDUs to be distributed.
Q_{tk}	Denotes a query k sent from site t.

ACC_{tk}	Subset of H to be accessed by a user identified with a site t $\in N$, and a query issued at t identified by an index k.
WRT_{sk}	Subset of H to be updated by a user identified with a site s $\in N$, and an update issued at s identified by an index k.
PAR_{tk}	Set of possible partition indexes for ACC_{tk}, where $p \in PAR_{tk}$ denotes the index corresponding to the choice of one partition or clustering configuration.
CLU_{tkp}	p^{th} partition of ACC_{tk}.
N_h	Number of clusters q that contain h $\in H$, over all queries.
A_{tk}^{pq}	Volume of data and messages obtained from cluster $q \in CLU_{tkp}$ after all the reducing operations have been performed.
A_{tk}^{pqr}	Sum of the volume of data and messages sent from cluster $q \in CLU_{tkp}$ to cluster $r \in CLU_{tkp}$ corresponding to the combined *critical* reducing operations from q to r.
$\bar{A}_{tk}^{pqr}$	Volume of data and messages sent from cluster $q \in CLU_{tkp}$ to cluster $r \in CLU_{tkp}$ corresponding to the *noncritical* reducing operation (if any) from q to r.
B_{sk}^{h}	Volume of data and messages sent from site s to update CRDU h $\in H$.

The following variables are to be evaluated:

Flow variables:

$x_s(i,j)$	Total message and data volume flow on arc (i,j) for queries and updates with s as the source site.

Allocation variables:

$$Y_i^h \quad = \begin{cases} 1 & \text{if h} \in \text{H is at i} \in \text{N} \\ 0 & \text{otherwise.} \end{cases}$$

Partition variables:

$$X_{t\,k}^{p} \quad = \begin{cases} 1 & \text{if } ACC_{tk} \text{ is partitioned according to} \\ & \text{configuration p of } PAR_{tk}, \\ 0 & \text{otherwise.} \end{cases}$$

Materialization variables:

$$X_{t\,k\,i}^{p\,q} \quad = \begin{cases} 1 & \text{if cluster q corresponding to partition p} \in PAR_{tk} \\ & \text{for query } Q_{tk} \text{ is accessed at node i} \\ 0 & \text{otherwise.} \end{cases}$$

The following notations are also used:

$$w_{ti} \quad = \begin{cases} 1 & \text{if node i} \in N \text{ is identical to node t,} \\ 0 & \text{otherwise.} \end{cases}$$

$$Y^h \quad = \sum_{j=1}^{|N|} Y_j^h$$

$$x_s(j,N) \quad = \sum_{i=1}^{|N|} x_s(j,i)$$

The *assembly flow value,* at any node i, corresponding to the total assembly flow with source node s is:

$$ASM_{si} = \sum_{Q_{tk}} \sum_{p \in PAR_{tk}} \sum_{q \in CLU_{tkp}} A_{t\,k}^{p\,q} X_{t\,k\,s}^{p\,q} \; [\, w_{si} - w_{ti} \,]$$

$$\textit{for all } s \in N,\ i \in N.$$

where $A_{t\,k}^{p\,q}\, X_{t\,k\,s}^{p\,q}$ is the assembly *outflow value* for source node s, and the assembly *inflow value* for sink node t (the user node). For any node which is neither s, the source node, nor t, the destination node, this flow value is equal to zero.

The *reducing flow value* at any node i, corresponding to the total file-to-file flow with source s is:

$$RDC_{si} = \sum_{Q_{tk}} \sum_{p \in PAR_{tk}} \sum_{q,\, r \in CLU_{tkp}} \sum_{j \in N, j \neq s} \left[A_{tk}^{pqr} + \overline{A}_{tk}^{pqr} \left(1 - X_{tkt}^{pr} \right) \right]$$

$$\times \left[w_{si} - w_{ji} \right] X_{tkj}^{pr} \, X_{tks}^{pq}$$

for all $s \in N,\ i \in N,$

The term $[\ 1 - X_{tkt}^{pr}\ ]$ is zero for any *noncritical* semi-join r[>q in the query processing sequence, and cluster r is located at user node t. It is equal to one otherwise. This means that cluster r need not be reduced any further since no other cluster outside t would benefit from this reduction. $X_{tki}^{pr}\ X_{tkt}^{pr} = 1$ if and only if i = t. Therefore $X_{tki}^{pr}\ X_{tkt}^{pr} = X_{tki}^{pr}\ w_{ti}$. Hence RDC_{si} can be rewritten as:

$$RDC_{si} = \sum_{Q_{tk}} \sum_{p \in PAR_{tk}} \sum_{q, r \in CLU_{tkp}} \sum_{j \in N, j \neq s} \left[A_{tk}^{pqr} + \overline{A}_{tk}^{pqr} \left(1 - w_{tj} \right) \right]$$

$$\times \left[w_{si} - w_{ji} \right] X_{tkj}^{pr} X_{tks}^{pq} .$$

The update flow value at any node i, corresponding to the total update flow with source s is:

$$UPD_{si} = \sum_{U_{sk}} \sum_{h \in WRT_{sk}} B_{sk}^{h} \left[Y^{h} w_{si} - Y_{i}^{h} \right]$$

for all $s \in N,\ i \in N.$

Following is the general optimization model:

$$\text{minimize} \quad Z = \sum_{(i,j) \in A} c_{ij} \sum_{s \in N} x_s(i,j),$$

subject to:

Flow conservation:

$$-x_s(i,N) + x_s(N,i) = -ASM_{si} - RDC_{si} - UPD_{si} \qquad \text{(mip-1)}$$

for all $s \in N,\ i \in N.$

Flow capacity:

$$0 \leq \sum_{s} x_s(i,j) \leq K_{ij} \qquad \text{(mip-2)}$$

for all $(i,j) \in A.$

Partitioning: Given a query Q_{tk}, a set of CRDUs to access, ACC_{tk}, only one clustering configuration $p \in PAR_{tk}$ must be selected.

$$\sum_{p \in PAR_{tk}} X_{t\,k}^{p} = 1 \qquad \text{(ip-1)}$$

for all Q_{tk}.

Materialization: Any cluster $q \in CLU_{tkp}$ must be associated with exactly one node in the network.

$$\sum_{i=1}^{|N|} X_{t\,k\,i}^{p\,q} = X_{t\,k}^{p} \qquad \text{(ip-2)}$$

for all $Q_{tk},\ p \in PAR_{tk},\ q \in CLU_{tkp}$.

Allocation: For each cluster, the CRDUs contained in it must be available.

$$\sum_{Q_{tk}} \sum_{p \in PAR_{tk}} \sum_{\substack{q \in CLU_{tkp} \\ where\ h \in q}} X_{t\,k\,i}^{p\,q} \leq N^h Y_i^h \qquad \text{(ip-3)}$$

for all $i \in N,\ h \in H.$

The mathematical model presented above is a (min-cost multi-commodity flow) linear mixed-integer program.

Example: Let the following CRDUs be defined with respect to the relations displayed in Appendix B.

h_1 = R1[NAME,AGE],

h_2 = R2[NAME,PHYSICIAN],

h_3 = R2[ORDER,NAME].

Let the target list of a query issued at site t be {NAME, AGE, PHYSICIAN, ORDER}. In the following notation the subscripts "tk" will be dropped since we are considering one user site and one query originated at that site. The set of CRDUs to access is: $ACC = \{h_1, h_2, h_3\}$. There are five possible clustering configurations for ACC, therefore $PAR = \{1, 2, 3, 4, 5\}$. The clustering configurations that ACC may assume are: $CLU_1 = \{<h_1>,<h_2>,<h_3>\}$, $CLU_2 = \{<h_1>,<h_2h_3>\}$, $CLU_3 = \{<h_1h_2>,<h_3>\}$, $CLU_4 = \{<h_1h_3>,<h_2>\}$, $CLU_5 = \{<h_1h_2h_3>\}$, where $<h_i\, h_j>$ stands for h_i *clustered with* h_j. In CLU_2, let $q = 1$ refer to $<h_1>$ and $q = 2$ refer to $<h_2h_3>$. For this query only, let t be the user site, let site 1 hold a copy of h_1 and site 2 hold a copy of h_2 and a copy of h_3 (Figure 5.2). Then $X_1^{21} = 1$, $X_2^{22} = 1$, $X_1^{22} = 0$, $X_2^{21} = 0$. The volume of messages and data A^{12} corresponds to the column $h_1[NAME]$, and the volume of messages and data A^{21} corresponds to one column $h_2[NAME] \bigcap h_3[NAME]$. Hence it is evident that because different possible clustering configurations of CRDUs are considered, and not the CRDUs individually, the problems of redundancy are taken care of, i.e. there is no need to send one column $h_2[NAME]$ and another column $h_3[NAME]$ to site 1. The volume of messages and data A^1 corresponds to $h_1[NAME,AGE]$ after the semi-join $h_1<NAME](h_2[NAME]h_3)$ occurs at site 1. Similarly, A^2 corresponds to $(h_2[NAME]h_3)[NAME,ORDER,PHYSICIAN]$ after the semi-join $h_1[NAME>(h_2[NAME]h_3)$ has taken place at site 2. •

When the flow capacity of the communication network is virtually infinite, constraints (mip-1) and (mip-2) are omitted and the objective for the *uncapacitated* model becomes:

$$\textit{minimize} \sum_{s \in N} \left\{ \sum_{Q_{tk}} \sum_{p \in PAR_{tk}} \sum_{q \in CLU_{tkp}} \left[\left(A_{t\,k}^{p\,q} X_{t\,k\,s}^{p\,q} \right.\right.\right.$$

$$+ \sum_{\substack{r \in CLU_{tkp} \\ r \neq q}} A_{t\,k}^{p\,qr}\, X_{t\,k\,t}^{p\,r}\, X_{t\,k\,s}^{p\,q} \left.\right) \overline{c}_{st}$$

$$+ \sum_{\substack{r \in CLU_{tkp} \\ r \neq q}} \sum_{\substack{i \in N \\ i \neq t}} \left[\left(A_{t\,k}^{p\,qr} + \overline{A}_{t\,k}^{p\,qr} \right) X_{t\,k\,s}^{p\,q}\, X_{t\,k\,i}^{p\,r} \right] \overline{c}_{si} \left.\right]$$

$$+ \sum_{U_{sk}} \sum_{h \in WRT_{sk}} \sum_{i \in N} \left(B_{s\,k}^{h}\, Y_{i}^{h} \right) \overline{c}_{si} \left.\right\},$$

subject to (ip-1), (ip-2) and (ip-3). In this case, the optimization model is a quadratic 0-1 integer program.

One might want to ask whether we need to include a constraint which states that two different FD-clusters in the same partition cannot be accessed at the same node to prevent the shipment of redundant data? For instance, if two CRDUs present in two different clusters located at the same site s, are submitted to semi-joins with CRDUs outside s, on an identical domain, say D, then values of D will be sent twice out of s (and into s). The answer to that question is that if an optimal solution is sought, more favorable partitions can be chosen. Therefore such an additional constraint needs not be enforced because it will be satisfied if it helps to decrease the cost. Similarly, a constraint which states that it is useless to consider a partition containing m clusters in a network with n nodes if m is larger than n is unnecessary. Indeed, the remark made above applies also to this case. It is nevertheless up to the designer to decide whether for some query Q_{tk} and its envelope ACC_{tk}, a partition p associated with a set of clusters $CLU_{t\,k}^{p}$ is useful or not.

The next chapter will be devoted to computational considerations and techniques to solve the large-scale optimization models proposed above.

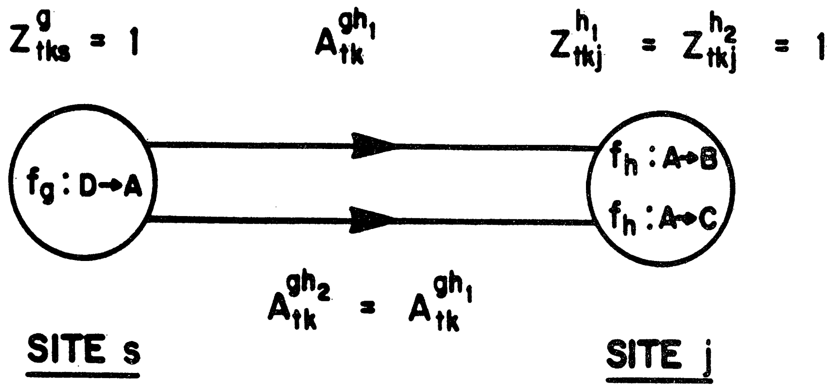


Figure 5.1. The placement of two redundant CRDUs at the same site

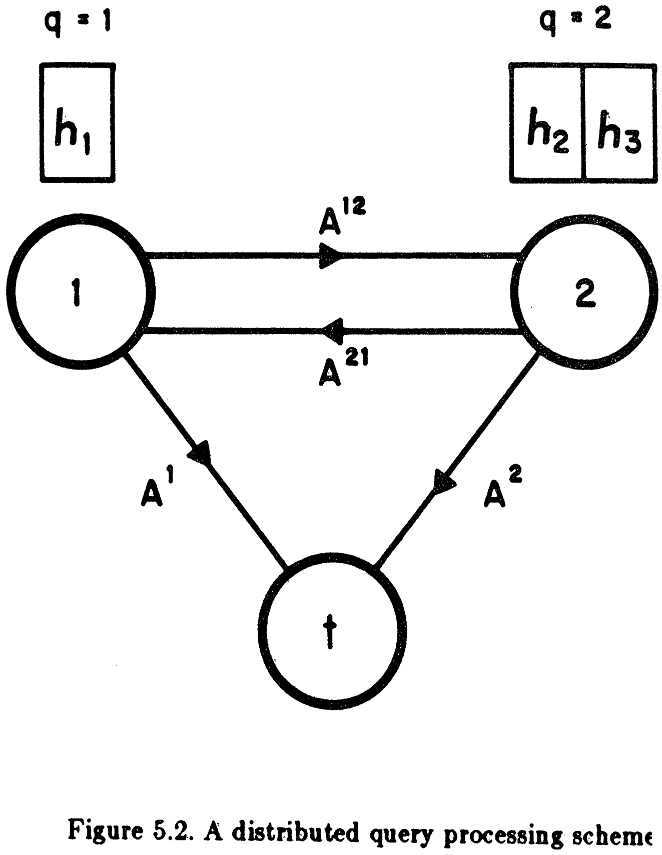


Figure 5.2. A distributed query processing scheme

CHAPTER 6

COMPUTATIONAL TECHNIQUES

When the constraints over the total flow of information on each channel, due to finite capacity limitations, are omitted, the mathematical model for the distribution of cross-referencing data units becomes a 0-1 integer program with quadratic cost and linear constraints. If the capacity constraints are included, the model assumes the form of a mixed integer program with a linear cost in real (flow) variables and with linear constraints that include both the real variables and the integer 0-1 variables. The first model will be referred to as the *uncapacitated* model, and the second model as the *capacitated* model. An algorithm which seeks an optimal solution, and an algorithm which systematically generates improved feasible solutions for the *uncapacitated* model will be described and discussed. Further, a heuristic technique which seeks near-optimal solutions for the capacitated model will also be described and discussed. Numerical results and comparisons will follow.

6.1. Solving the uncapacitated model by integer programming

If plenty of computer money and time is available, and if the size of the problem permits, then it may be desirable, and indeed it is possible, to find an optimal solution. Classical integer programming techniques such as *Branch and Bound* (BNB) can be used to accomplish that goal. In the next section we show that the BNB method can be applied conveniently because only a subset of the 0-1 variables actually need to be considered for branching. The method is then applied successfully to a 350 variables problem in Section 4.1.2. In Section

4.1.3 it is explained why more efficient techniques, even though non optimal, must be considered.

6.1.1. Description of a branch-and-bound algorithm

For some not-so-large problems, optimal integer programming techniques can be applied on the *uncapacitated* model.

It is now shown that there exists some dependency among the 0-1 variables of the uncapacitated model. The results of the following proposition will be used in the application of the BNB algorithm.

Proposition 17:

Let $X^{*} = (\cdots X_{tki}^{pq*} \cdots ; \cdots Y_i^{h*} \cdots ; \cdots X_{tk}^{p*} \cdots)$ *be an optimal solution of the uncapacitated model, such that all the* Y_i^{h*} *and* X_{tk}^{p*} *variables are 0 or 1, and such that all the variables* X_{tki}^{pq*} *satisfy:* $0 \leq X_{tki}^{pq*} \leq 1$; *then there exists an optimal solution* $\overline{X} = (\cdots \overline{X}_{tki}^{pq} \cdots ; \cdots Y_i^{h*} \cdots ; \cdots X_{tk}^{p*} \cdots)$ *such that* $\overline{X}_{tki}^{pq} = 0$ *if* $X_{tki}^{pq*} = 0, \overline{X}_{tki}^{pq*} = 1$ *if* $X_{tki}^{pq*} = 1$ *and* $\overline{X}_{tki}^{pq} = 0$ *or* 1 *if* X_{tki}^{pq*} *is fractional.*

proof: Suppose that, under the conditions of the proposition, there exists at least one Q_{tk} , p and q , and some $s_1, s_2, \ldots, s_n \in N$ such that: $X_{tks_1}^{pq}$, $X_{tks_2}^{pq}, \ldots, X_{tks_n}^{pq}$ are all different from zero, and constraints (ip-1), (ip-2) and (ip-3) are satisfied, and such that the cost is optimal. The expression for the cost contains a term of the form:

$$A_{tk}^{pq}\overline{c}_{ts_1}X_{tks_1}^{pq} + \cdots + A_{tk}^{pq}\overline{c}_{ts_n}X_{tks_n}^{pq} + \cdots$$

$$+ \sum_{r \neq q}\left[A_{tk}^{pqr}\overline{c}_{ts_1}X_{tkt}^{pr}X_{tks_1}^{pq} + \cdots + A_{tk}^{pqr}\overline{c}_{st_n}X_{tkt}^{pr}X_{tks_n}^{pq}\right]$$

$$+ \sum_{j}\sum_{r \neq q}\left[(A_{tk}^{pqr} + \overline{A}_{tk}^{pqr})\,\overline{c}_{js_1}X_{tkj}^{pr}X_{tks_1}^{pq} + \cdots + (A_{tk}^{pqr} + \overline{A}_{tk}^{pqr})\,\overline{c}_{js_n}X_{tkj}^{pr}X_{tks_n}^{pq}\right]$$

$$= \sum_{s_i} \left\{ \left[A_{t\,k}^{p\,q} + \sum_{r} A_{t\,k}^{p\,qr} X_{t\,k\,t}^{p\,r} \right] \bar{c}_{ts_i} + \sum_{j} \sum_{r} \left(A_{t\,k}^{p\,qr} + \overline{A}_{t\,k}^{p\,qr} \right) X_{t\,k\,j}^{p\,r} \bar{c}_{js_i} \right\} X_{t\,k\,s_i}^{p\,q}$$

$$\stackrel{\Delta}{=} \sum_{s_i} E^{i} X_{t\,k\,s_i}^{p\,q}.$$

Let $s^* \in \{ s_1, s_2, \ldots, s_n \}$ be such that:

$$A_{t\,k}^{p\,q} + \sum_{r} \left(A_{t\,k}^{p\,qr} + \overline{A}_{t\,k}^{p\,qr} \right) X_{t\,k\,t}^{p\,r} c_{ts^*} + \sum_{j} \sum_{r} A_{t\,k}^{p\,qr} X_{t\,k\,j}^{p\,r} c_{js^*} \stackrel{\Delta}{=} E^*$$

and $E^* \le E^i$ for all $i = 1, \cdots, n$. Then, let us set: $X_{t\,k\,s^*}^{p\,q} = 1$ and $X_{t\,k\,s_h}^{p\,q} = 0$ for any $s_h \in \{ s_1, \ldots, s_n \} - \{s^*\}$. Equations (ip-1), (ip-2) and (ip-3) are still satisfied after the substitution. Yet, $\sum_{s_i} E^i X_{t\,k\,s_i}^{p\,q} \ge \sum_{s_i} E^* X_{t\,k\,s_i}^{p\,q} = E^* \sum_{s_i} X_{t\,k\,s_i}^{p\,q} = E^* = E^* X_{t\,k\,s^*}^{p\,q}$.

Therefore after the substitution a solution is obtained that is at least as good as the previously found solution. This type of substitution can be extended to other variables of the type $X_{t\,k\,i}^{p\,q}$ that are still fractional. •

The BNB algorithm written in pseudocode follows. It is self-explanatory. The branch-and-bound technique of partial enumeration has been reported in [MURT 76, SALK 75, CHEN 80].

ALGORITHM Branch_And_Bound:

```
    Incumbent := ∅ /* Current problem feasible all integer */
    List := { $CP^0$ } /* List of candidate problems */
    Branch
    if Incumbent = ∅ then NoSolution
    else Optimal:=Incumbent

end Branch_And_Bound.
```

ALGORITHM Branch:

```
    while List≠ ∅ do
    begin
       List := List - { $CP^i$ }
       /* Use a LIFO search */
       $CP^{i0}$ := Branch0($CP^i$)
       /* Set a fractional variable to 0 */
```

```
        $CP'^1$ := Branch1($CP'$)
        /* Set it to 1 */
        $LB'^0$ := LowerBound($CP'^0$)
        $LB'^1$ := LowerBound($CP'^1$)
        UpdateIncumbent($CP'^0$)
        $K_1$ := Dangle
        UpdateIncumbent($CP'^1$)
        if (~ $K_1$ and ~ Dangle) then return
        /* $CP'^1$ is still good */
        if ~ $K_1$ then
        begin
            List := List $\bigcup$ { $CP'^1$ }
            Branch
        end
        else begin
            if ~ Dangle then /* $CP'^0$ still good */
            begin
                List := List $\bigcup$ { $CP'^0$ }
                Branch
            end
            else begin /* $CP'^0$ & $CP'^1$
                are still good for further branching */
                if $LB'^0 < LB'^1$ then
                begin
                    List := List $\bigcup$ { $CP'^1$ }
                    List := List $\bigcup$ { $CP'^0$ }
                end
                else begin
                    List := List $\bigcup$ { $CP'^0$ }
                    List := List $\bigcup$ { $CP'^1$ }
                end
                 Branch
                 Branch
            end
        end
    end

end Branch.
```

ALGORITHM UpdateIncumbent (CP'):

```
    LBIncumbent := $\infty$
    Dangle := true
    $LB'$ := LowerBound($CP'$)
    if Fathom then /* $CP'$ is solved all integer */
    begin
        if $LB' <$ LBIncumbent then
        Incumbent := $CP'$ /* Update incumbent */
        /* also prune the CP's in List that have higher cost */
        Dangle := false
        return
    end
    if Infeasible then /* No more branching */
        Dangle := false
    if $LB^i \geq LBIncumbent$ then Dangle := false
    /* No need to branch since no hope of improving
     incumbent along that branch */

end UpdateIncumbent.
```

Note that in accordance with Proposition 17, only the variables of type Y_i^k and $X_{t\,k}^p$ are considered for possible branching; i.e. they are the only ones which are deliberately fixed to 0 or 1 in the BNB algorithm. This constitutes a numerical asset as will be shown below in Section 4.1.2 where a problem which normally requires 350 0-1 variables needs only be branched over 99 of those.

To compute the cost of a candidate problem, (*LowerBound*(CP^i)), the choice of a method for large scale optimization is crucial, because in the *uncapacitated* model, the number of variables can easily exceed one thousand. It appears that algorithms based on *gradient projection* methods and *reduced gradient* methods [HIMM 72], are best suited to our model because they take advantage of the linearity of the constraints; for that reason the optimization package MINOS [MUR 77, MUR 78] which is based on such techniques was chosen. MINOS is a program that solves large scale, bounded variable, linear or nonlinear programming problems with linear constraints. It performs very well and can handle models with several thousand variables.

The BNB algorithm described above has been implemented in FORTRAN. The *Lower Bound* procedure is realized with a call to MINOS. One problem was solved numerically. The results are discussed in the next section.

6.1.2. A practical problem

A practical problem which can be modeled using the *uncapacitated* model is solved optimally. The MRDB of Appendix C is continued (see also Section 4.7). Although there is no constraint as to the amount of information flow in the network, the organization wishes to distribute the data that will be subject to scheduled transactions in such a way that the total communication cost is minimized over all the queries and updates issued at all the sites. The user-provided FDs are listed in Table C.1, but after the application of Algorithm *Reduce* (Section 4.7) an LR-minimum cover of the FDs is as shown in Table 6.1. Attribute TEST includes the TEST_TYPE and CODE_NUMBER and the TECHNIQUE used. RESULT includes the actual numerical result for a test, the PRECISION, the ACCURACY, the COST and the amount of TIME spent. The types of queries and updates which are commonly asked are listed in a simplified form in Table 6.2. Table 6.3 shows what queries and updates are being initiated at each site. Each query and update index is followed by the corresponding

Functional Dependencies for MRDB		
ORDER	$\longrightarrow$	SERVICE
ORDER	$\longrightarrow$	TEST
ORDER	$\longrightarrow$	PATIENT_NO
ORDER	$\longrightarrow$	DATE
TEST	$\longleftrightarrow$	TEST_TYPE,CODE_NO,TECHNIQUE
TEST,DATE,PATIENT_NO	$\longrightarrow$	RESULT
RESULT	$\longleftrightarrow$	TOTAL,PRECISION,ACCURACY,COST,TIME
PATIENT_NO	$\longrightarrow$	NAME
PATIENT_NO	$\longrightarrow$	AGE
SERVICE	$\longrightarrow$	PHYSICIAN
PHYSICIAN	$\longrightarrow$	WARD
NAME	$\longrightarrow$	WARD

Table 6.1. Some functional dependencies for the MRDB.

estimated frequency (preceded by a "×" sign) expressed in 1 / *minutes* units. For instance, site 4 which stands for the main hospital laboratory, is only concerned with queries 4, 6, 9 and 7, and is the only site to perform updates on the MRDB. After processing the initial FDs as discussed in Section 4.4.4 (see also Section 4.7), each query is assigned an envelope of CRDUs as shown in Table 6.4. The derived FDs are denoted $h_1, \ldots, h_{13}$, or simply 1,...,13. At this point it is possible to calculate a distributed query processing sequence for each possible clustering situation of the CRDUs belonging to the envelope of a specific query. (See steps (14) and (15) of Algorithm *Select* in Section 4.6 and 4.7.) This is displayed in Table 6.5 where only the information pertinent to site 2 is shown: Query 3 needs to access CRDUs 4, 5, and 6. There are five possible clustering situations to be considered (Table 6.5 column 3). The

Typical scheduled queries for MRDB		
Number	Given attributes	Target attributes
1	ORDER	PHYSICIAN,WARD
2	ORDER	RESULT
3	DATE,PATIENT_NO,TEST	ORDER
4	WARD	PATIENT_NO
5	ORDER	NAME,WARD
6	PHYSICIAN	ORDER,PATIENT_NO
7	NAME,DATE,TEST	RESULT
8	ORDER	SERVICE
9	PHYSICIAN	SERVICE,PATIENT_NO

Typical scheduled updates	
Number	New attributes
1	ORDER,TEST,RESULT,DATE
2	PATIENT_NO,NAME,AGE

Table 6.2. Typical user-provided transactions for the MRDB.

Usage of MRDB transactions		
Site	Query	Update
1	$1 \times 2, 2 \times 1, 3 \times 1, 5 \times 1, 8 \times 1$	none
2	$1 \times 1, 3 \times 2, 9 \times 1$	none
3	$5 \times 3, 7 \times 4, 8 \times 4$	none
4	$4 \times 1, 6 \times 1, 9 \times 1, 7 \times 3$	$1 \times 1, 2 \times 2$

Table 6.3. Typical frequencies for the MRDB transactions.

CRDUs accessed for query processing		
Query	CRDUs	Possible partitions of CRDUs

1	h1=ORDER→PHYSICIAN
	h2=PHYSICIAN→WARD
		<h1,h2>
		<h1><h2>

2	h3=ORDER→RESULT	<h3>

3	h4=ORDER→TEST
	h5=ORDER→DATE
	h6=ORDER→PATIENT_NO
		<h4, h5, h6>
		<h4><h5><h6>
		<h4, h5><h6>
		<h4><h5, h6>
		<h4, h6><h5>

4	h7=PATIENT_NO→WARD	<h7>

5	h8=ORDER→NAME
	h9=NAME→WARD
		<h8><h9>
		<h8, h9>

6	h1=ORDER→PHYSICIAN
	h6=ORDER→PATIENT_NO
		<h1><h6>
		<h1, h6>

7	h10=PATIENT_NO,DATE,TEST→RESULT
	h11=PATIENT_NO→NAME
		<h10><h11>
		<h10, h11>

8	h12=ORDER→SERVICE	<h12>

9	h13=SERVICE→PHYSICIAN
	h12=ORDER→SERVICE
	h6=ORDER→PATIENT_NO
		<h13, h12, h6>
		<h13><h12><h6>
		<h13, h12><h6>
		<h13><h12, h6>
		<h13, h6><h12>

Table 6.4. All the FDs selected for distribution, and the possible partitioning configurations, for the MRDB.

matrix $[A_t^{q_k r}]$ (Table 6.5 column 5) is obtained, for each user's site-query couple, after running a distributed query processing algorithm for each possible clustering situations of CRDUs, and after adding up all the data volume flows taking place between each couple of CRDU clusters. The entries are in *kilobits* . All the user's informations are inputted to the

BNB algorithm in the form shown in Appendix D.

The MRDB problem considers a network of 4 sites, with 3 to 5 typical scheduled query types per site, 2 update types, and a total of 13 CRDUs to be distributed. The *uncapacitated* version of the model may be used. The total number of 0-1 variables turns out to be 350 for this problem. However, as was explained in the preceding section, since only the variables of the types Y_t^k and X_{tk}^p , namely the *allocation* and the *partition* variables, need to be considered by the BNB algorithm, the total number of variables is decreased to 99. There are a

Query profile of site 2						
Query index	CRDUs	Partitions	Clusters	Data volume flow		
k	ACC_{2k}	$p \in PAR_{2k}$	$q \in CLU_{2k}^p$	$[A_{2k}^{qr}]$		
1	1, 2	1={<1, 2>}	1=<1, 2>	450	-	-
		2={<1>, <2>}	1=<1>	400	5	-
			2=<2>	4	10	-
3	4, 5, 6	1={<4, 5, 6>}	1=<4, 5, 6>	200	-	-
		2={<4>, <5>, <6>}	1=<4>	600	0	0
			2=<5>	0	800	0
			3=<6>	200	200	200
		3={<4, 5>, <6>}	1=<6>	600	0	-
			2=<4, 5>	200	200	-
		4={<4>, <5, 6>}	1=<4>	500	0	-
			2=<5, 6>	200	200	-
		5={<5>, <4, 6>}	1=<5>	200	200	-
			2=<4, 6>	0	800	-

Table 6.5. An example of numerical input (corresponding to site 2 in the MRDB problem) to be provided to the distribution program.

total of 132 constraints for this problem, not including the constraints of the type $0 \leq X \leq 1$ where X is any of $X_{t\,k\,l}^{p\,q}$, Y_{l}^{h} or $X_{t\,k}^{p}$. The BNB algorithm was run with MINOS on the Amdahl 470V/8 of the Michigan Terminal System to solve this problem. A final 0-1 optimal solution was obtained in approximately 4 minutes* of CPU time after branching 22 times. The optimal total communication flow was found to be 3390 *kilobits/hour*, and the corresponding configuration of the CRDUs is as shown in Table 6.6, for a site 1 and its queries, as given by the allocation variables. The non-zero *materialization* variables indicate the optimal way to access the CRDUs for that query from the selected site (Table 6.6). For query 5, it was found that $X_{1\,5}^{1} = X_{1\,5\,4}^{1\,1} = 1.$ rather than $X_{1\,5}^{2} = 1$ found in the present solution, there would have been no difference in communication cost since the total assembly flow is equal to 100 *kilobits/hour* in both cases. Furthermore, CRDU <8> which stands for (<ORDER → NAME>) sends no data-volume to the user node (node 1), whereas CRDU <9> (<NAME → WARD>) sends 100 *kilobits/hour* of data-volume to the user site; therefore, although site 1 would be expected to get access to two clusters <8> and <9> at different sites (refer to the remark, made in Section 5.2, and to the example also in Section 5.2 associated with Figure 5.1), in this particular example it is irrelevant wether CRDUs <8> and <9> are accessed in one common relational table or two distinct relational tables because the assembly flow is identical in both cases. (Of course, if CRDUs <8> and <9> were positioned in distinct sites, we would have to account for an additional 100 *kilobits/hour* of reducing flow.)

6.1.3. Practical limitations

The BNB algorithm allows the finding of an optimal solution for the *uncapacitated* model. However, there are well known disadvantages:

*The package MINOS was available as a stand-alone program only, therefore the BNB algorithm was run semi-manually. The total CPU time was obtained after compensating for repetitious calls to various initialization routines and input-output operations.

Solution for the MRDB problem and the input data of Appendix D. Only the results relevant to user site 1 are shown.				
k	ACC_{1k}	PAR_{1k}	CLU^{p}_{1k}	site
1	{1, 2}	2 = {<1>, <2>}	1 = <1> 2 = <2>	1 2
$X_{1}{}^{2}{}_{1} = 1$ $X_{1}{}^{2}{}_{1}{}^{1}{}_{1} = X_{1}{}^{2}{}_{1}{}^{2}{}_{2} = 1$ $Y_1^1 = Y_2^2 = 1$				
2	{3}	1 = {<3>}	1 = <3>	1
$X_{1}{}^{1}{}_{2} = 1$ $X_{1}{}^{1}{}_{2}{}^{1}{}_{1} = 1$ $Y_1^3 = 1$				
3	{4, 5, 6}	1 = {<4, 5, 6>}	1 = <4, 5, 6>	4
$X_{1}{}^{1}{}_{3} = 1$ $X_{1}{}^{1}{}_{3}{}^{1}{}_{4} = 1$ $Y_4^4 = Y_4^5 = Y_4^6 = 1$				
5	{8, 9}	2 = {<8>, <9>}	1 = <8> 2 = <9>	4 4
$X_{1}{}^{2}{}_{5} = 1$ $X_{1}{}^{2}{}_{5}{}^{1}{}_{4} = X_{1}{}^{2}{}_{5}{}^{2}{}_{4} = 1$ $Y_4^8 = 1$ $Y_4^9 = 1$				
8	{12}	1 = {<12>}	1 = <12>	4
$X_{1}{}^{1}{}_{8} = 1$ $X_{1}{}^{1}{}_{8}{}^{1}{}_{4} = 1$ $Y_4^{12} = 1$				

Table 6.6. A partial solution for the MRDB problem.

(1) The classical file allocation problem is np-complete [ESWA 74], so is, a fortiori, the allocation of CRDUs [APER 80]. In practice the BNB algorithm may not be able to

overcome the curse of cardinality.

(2) Even if an optimal solution can be arrived at with BNB for a specific problem, the price to be paid in CPU time and computer money may be unacceptable.

Therefore, although the BNB algorithm remains a possible option to fully solve the *uncapacitated* model, clearly, other less expensive and faster algorithms must be considered.

In the next section, such an algorithm is described and applied to the MRDB problem. The final section of this chapter deals with the solution of the *capacitated* model.

6.2. Improving feasible solutions for the uncapacitated model

Since it is not always feasible to seek an optimal solution to the *uncapacitated* model, a more efficient, although non optimal, algorithm is proposed to generate "good" feasible solutions for the *uncapacitated* model. How good those suboptimal solutions are will be discussed shortly.

The forthcoming Algorithm *XrefU* is based on an *adaptive* search method developed in [HOLL 75]. This algorithm can be applied to optimize a performance function evaluated over a set of character strings with finite length and fixed representation. In particular, it can be applied to unconstrained 0-1 integer programming models where a character string corresponds to a vector of 0's and 1's, and the performance function is mapped to the cost (in our case, the communication cost). Since the theory of adaptive algorithms has been studied elsewhere [HOLL 75], the desirable properties that adaptive algorithms exhibit will be reiterated without proof: (1) They can handle problems that feature high dimensionality and many local optima. Says [HOLL 75]:

> The algorithm's power is most evident when it is confronted with problems involving high dimensionality (hundreds to hundreds of thousands of attributes, as in genetics and economics) and multitudes of local optima.

(2) They feature intrinsic parallelism in the sense that the solution space is searched uniformly, and there cannot be a terminal entrapment on a local optimum. (3) They feature the

quality of a robust search in the sense that the number of trials allocated to the observed best solutions increases exponentially with respect to the remainders.

The adaptive algorithm maintains a list of strings of finite length, called *structures*. String manipulation operations are used to improve the list such that the 0's or 1's (or combination thereof), at specific positions, which tend to improve the performance of structures, will persist consistently within the structures to appear in subsequent lists, whereas the 0's or 1's which tend to deteriorate the performance of structures, will eventually recede. [HOLL 75] is entirely devoted to the treatment of adaptive algorithms.

If the structures are simply strings of 0's and 1's of length l, then any such string with an arbitrary arrangement of 0's and 1's may appear in the list (after a repetitive application of the string manipulation operations) at one stage in the execution of the algorithm, and thus constitutes an acceptable structure. Obviously, this is not satisfactory in the context of solving a constrained integer optimization program of the type featured by the *uncapacitated* model. Indeed if a structure is mapped to a vector of 0-1 variables, then only those vectors that are feasible would have to be considered. Furthermore, if the list to be maintained by the adaptive algorithm were to contain feasible solution vectors only, there would be no guarantee that a repetitive application of the string manipulation operations on the 0-1 strings which represent those vectors, would not interfere with feasibility. Therefore, when solving the *uncapacitated* model, the 0-1 strings of the list cannot simply be mapped in a one-to-one manner to vectors of 0-1 variables in the mathematical model. The structures of the list must be defined in such a way that they represent feasible solutions (in an encoded form), and such that they remain meaningful at all time in the course of the execution of the algorithm, when submitted to any number of applications of the string manipulation operations. The adaptive algorithm used to solve the *uncapacitated* model is described in the next section.

6.2.1. Adaptive random search for the uncapacitated model

It is shown how feasible solutions of the *uncapacitated* model are encoded into structures which can be used by an adaptive random search algorithm.

Any element of the list maintained by the adaptive algorithm will be a string of 0's and 1's of length v, such that it can be decoded and mapped to the variables $x(1), \ldots, x(m)$ of the *uncapacitated* model. (These variables correspond to the components of the vector $(\cdots X_{t\,k\,l}^{p\,q} \cdots ; \cdots Y_t^h \cdots ; \cdots X_{t\,k}^p \cdots)$.) The code for a structure is as follows: A structure is divided into *detectors* (numbered 1 through w, where w is equal to the total number of couples (t,k) in the mathematical model). Each detector has a fixed width (in *bits*) which can either be equal to a constant *width*1, *width*2 or *width*3; these constant widths are sufficient to represent a partition index of CRDUs with respectively, one, two or three clusters of CRDUs*. *Width*1 is defined to be the number of bits to be allocated to a detector which corresponds to a query that needs to access only one CRDU. Therefore *width*1 is simply equal to the number of bits required to identify one node in the computer network, when the nodes are numbered in binary form (the length, in bits, of a node binary representation is referred to as *nodesize*). For instance, if there are four nodes in the computer network, then *width*1 is equal to 2 *bits*. *Width*2 is defined to be the number of bits to be allocated to a detector which corresponds to a query that may access up to two distinct CRDUs. Therefore *width*2 is two *nodesize* bits long (to account for up to two nodes to be accessed) plus one bit to indicate the partition index (i.e if the two CRDUs are clustered at one node or located at different nodes). Thus, for a four node network, *width*2 is equal to five. Similarly *width*3 is defined for queries which may access up to three distinct CRDUs. Therefore *width*3 is three *nodesize* bits long plus three bits (to indicate the partition index which can be 1 through 5). Thus, for a four nodes network *width*3 is equal to 9. Figure 6.1 contains some useful notations. In Figure 6.3-i, a structure with twelve site-query couples is shown. The encoding scheme is indicated in Fig-

*Obviously, if the simultaneous access of more than three CRDUs is allowed at a time, for any query, then the ure 6.2. In Figure 6.3-i the first row of the structure contains the partition indexes for each query. The second, third and fourth rows contain respectively the node indexes for the first, the second and the third cluster for each partition. Each column corresponds to a detector as defined earlier. Note that queries Q_1 and Q_3 have only one CRDU in their envelope ($m = 1$). Queries Q_2, Q_4, and Q_6 have two CRDUs in their envelopes ($m = 2$) and so on. Figure 6.3-ii depicts the binary representation of the same structure, following the encoding indicated in Figure 6.2-ii. Thus, query Q_1 which has one possible partition and one cluster, is only represented by the node at which the cluster is located, which for this particular case is node 4 (11 in binary form). Similarly, the node to which the unique cluster in CLU_3^1 of query Q_3 is assigned, is node 3 (10 in binary form). Next, query Q_2 is shown to be solved through the access of its envelope in the form of partition 1. Thus, $p_2 \in PAR_2$ is equal to 1 (0 in binary form). This particular partition of ACC_2 results in CLU_2^1. The unique cluster in CLU_2^1 is shown to be accessed at site 2 (01 in binary form). Since there is no other cluster in CLU_2^1, therefore the entry for the node of cluster 2 is λ which means "don't care". Other detectors are similarly represented.

An algorithm to decode a structure and extract the actual values of its corresponding solution vector follows:

ALGORITHM EvalX:

```
$Y_i^h := 0$
for all $Q_{tk}$ do
begin
    $p := decodepar(\tau_i(Q_{tk}))$
    $X_{tk}^p := 1$
    for all $p' \in PAR_{tk} : p' \neq p$ do
    begin
        $X_{tk}^{p'} := 0$
        for all $q \in CLU_{tk}^{p'}$ do
        begin
            for all $i \in N$
                $X_{tki}^{p'q} := 0$
```

present representation needs to be extended accordingly.

```
        end
      end
      for all $q \in CLU^{p}_{t\,k}$ do
      begin
        $r := decodenode(\rho_t\,(q,p))$
        $X^{p\,q}_{t\,k\,r} := 1$
        for all $h \in q$ do $Y^{h}_{r} := 1$
        for all $j \neq r$ do
        $X^{p\,q}_{t\,k\,j} := 0$
      end
    end
```

end EvalX.

Using this decoding technique, the structures can be submitted to the string manipulation operations without fear of interfering with the feasibility of solutions. Indeed, any string of bits of length v, where

$$v = \sum_{i=1}^{w} width(i) \qquad width(i) \in \{width1, width2, width3\},$$

maps to a unique feasible solution to the *uncapacitated* model.

$$r \overset{\Delta}{=} (t,k) \textit{ a site-query couple}$$

$$\{Q_r \; : |ACC_r| \leq m\} \overset{\Delta}{=} Q^m$$

$$Q_r \in Q^m \rightarrow PAR_r \approx P^m \overset{\Delta}{=} \{1,2, \cdots ,5\} \quad (m = 3),$$

$$\tau_i : Q^m \rightarrow P^m \textit{ random assignment.}$$

$$p \in P^m \rightarrow CLU_r^p \approx R^m \overset{\Delta}{=} \{1,2,3\} \quad (m = 3),$$

$$\rho_i : R^m \times P^m \rightarrow N \bigcup \{\lambda\} \textit{ random assignment.}$$

$$\lambda \equiv \textit{don' t care}$$

Figure 6.1. Random search assignments.

<STRUCTURE>	::=	<PORTION-1><PORTION-2><PORTION-3>
<PORTION-m>	::=	<SUBSTR-m>|∅
<SUBSTR-m>	::=	<DETEC-m>|<SUBSTR-m><DETEC-m>
<DETEC-1>	::=	<NODE>
<DETEC-2>	::=	<PARTITION-2><NODE><NODE>
<PARTITION-2>	::=	0|1
<DETEC-3>	::=	<PARTITION-3><NODE><NODE><NODE>
<PARTITION-3>	::=	000|001|010|100|110|101|011|111
<NODE>	::=	00|01|10|11

$$|<STRUCTURE>| = |Q^1| \times nodesize + |Q^2| \times (2nodesize + 1)$$

$$+ |Q^3| \times (3 + 3nodesize) \overset{\Delta}{=} v$$

$$|Q^1| + |Q^2| + |Q^3| \overset{\Delta}{=} w$$

[I]

Code for a structure when |N| = 4 and |ACC| <= 3

0-1 representation of a <PARTITION-m> field in a detector			
m	Partition index $p = \tau_i(Q)$	Binary representation of <PARTITION-m>	partitioning
2	1	0	<AB>
	2	1	<A><B>
3	1	001	<ABC>
	2	010 or 110	<A><BC>
	3	011 or 111	<AB><C>
	4	100 or 000	<AC><B>
	5	101	<A><B><C>

[II]

Structure definition for Algorithm XrefU when N = { 1, 2, 3, 4 }

Figure 6.2. Encoding scheme for the adaptive algorithm.

At step i . . .

	m = 1		m = 2			m = 3						
$Q_r =$	1	3	2	4	6	5	7	8	9	10	11	12
$p_r = r_i(Q_r) =$	1	1	1	2	2	3	2	1	4	5	4	3
$\rho_i[1,p_r] =$	4	3	2	1	3	2	1	3	4	1	4	3
$\rho_i[2,p_r] =$			λ	3	4	2	1	λ	3	1	4	2
$\rho_i[3,p_r] =$						λ	λ	λ	λ	2	λ	λ

[i]

structure

| 1 1 1 0 0 0 1 - - 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 - - 0 1 0 0 0 0 0 - - . . . |

[ii]

physical binary representation

$N = \{1, 2, 3, 4\}$

Figure 6.3. **A structure for the adaptive algorithm.**

The adaptive algorithm used to solve the model follows shortly. At any step t in the execution of the algorithm, a list of u structures (*structure*[.]) is maintained. *Perf*[*i*] denotes the performance of structure *structure*[*i*]. Each structure is decoded using *EvalX* shown earlier, and the resulting vectore is plugged into a cost-evaluation routine. Typically, the performance decreases when the cost increases. *Ptercur* and *ptercurdown* are two pointers in the current list of structures at step t. *Pternext* is a pointer in the new list *strucnext*[.], which will become the current list at step $t + 1$. Procedure *EvalOffspring* evaluates the offspring of each structure in the current list. *Offspring*[*i*] is defined to be an integer function of *perf*[*i*] / *perf_ave* where *perf_ave* is the average performance for the current list. If *Offspring*[*ptercur*] is non-zero, then *structure*[*ptercur*] participates in string-manipulation operations (*OperateLeft, OperateRight*) with another structure *structure*[*ptercurdown*] to produce one or several new structures *strucnext*[*pternext*] in the new list. The string- manipulation operations include *mutation, inversion* and *cross–over* as defined in [HOLL 75]. They are summarized in Figure 6.4. Algorithm *XrefU* follows:

ALGORITHM XrefU: /* Adaptive random search to solve the *uncapacitated* model */

```
t:=0
ReadList /* Arbitrary list of structures */
for t≤tmax do /* Termination criterion */
begin
 for i=1 to u do
  perf[i]:=eval(structure[i]) /* Performance of each structure */
 t:=t+1
 ptrnext:=0
 EvalOffspring /* Calculate the offspring of each structure
 as a function of its performance */
 ptercur:=1
 while (ptercur≤u) & (ptrnext<u) do
 begin
  ptercurdown:=ptercurdown+1
  while  offspring[ptercur]≠0 & ptrnext<u
   & ptercurdown≤u do
  begin
   if offspring[ptercurdown]≠0 then
   begin
    offspring[ptercurdown]:=offspring[ptercurdown]-1
    offspring[ptercur]:=offspring[ptercur]-1
```

```
            ptrnext:=ptrnext+1
            strucnext[ptrnext]:=OperateLeft(structure[ptercur],
            structure[ptercurdown])
            strucsecond:=OperateRight(structure[ptercur],
            structure[ptercurdown])
            if ptrnext<u then
            begin
              ptrnext:=ptrnext+1
              strucnext[ptrnext]:=strucsecond
            end
          end
          else if ptercurdown=u then
          begin
            ptrnext:=ptrnext+1
            strucnext[ptrnext]:=Operate(structure[ptercur])
            offspring[ptercur]:=offspring[ptercur]-1
          end
          ptercurdown:=ptercurdown+1
        end
        ptercur:=ptercur+1
      end
      if ptrnext<u then for i=ptrnext+1to u do
      strucnext[i]:=structure[u]
      structure:=strucnext
    end

end XrefU.
```

1.1.1. Solving the MRDB problem with XrefU

Some properties of Algorithm *XrefU* are listed.

Algorithm *XrefU* may be initiated provided that an initial list of structures (encoded feasible solutions) is readily available. The length of this initial list is arbitrary, and so is its content; i.e. any structures may be used to fill it in. The longer the list is, the more information is generated in the course of the execution of the algorithm, however the performance of the algorithm will tend to degrade. Algorithm *XrefU* was implemented in PL/1. Generally, it was found that the list may contain between 15 and 40 structures for best overall results. Five runs were conducted, for the MRDB problem, on the Honeywell 870 of the Rome Air Development Center (via the Multics system). Each run was associated with a different, and completely arbitrary, list of structures. The results are shown in Figure 6.5.

Although the results obtained are not optimal, several positive remarks can be made: (1) The algorithm quickly eliminates the feasible solutions which are too expensive. (2) Even when the list is only 20 structures long, the algorithm is still capable of improving solutions at a steady rate within an average of 21 iterations, or an estimated 6.69 *seconds* of the Amdahl. Stagnation signs may only appear after the 30^{th} iteration, or so. Refer to remark (5) for possible improvements. (3) The best cost obtained was 4360 *kilobits/hour*, observed after the 47^{th} iteration of run 1 (which used a list of 34 structures) after an estimated 24.15 *seconds* of the Amdahl. This is relatively close to the optimal cost of 3390 *kilobits/hour* since the lowest cost in the initial list was observed to be 12,900 *kilobits/hour*, and the highest cost was 52,200 *kilobits/hour*. Also, as can be shown in Figure 6.5, the difference between the optimal cost and the best cost obtained with *XrefU* is 970 *kilobits/hour*, which is of the order of magnitude corresponding to the shipment of the volume of data for one query. (4) Although the best feasible solution observed with *XrefU*, after a reasonable amount of CPU time, is not likely to be the true optimal solution, *XrefU* generates low cost 0-1 feasible solutions that can be used to a great advantage by a more conventional BNB algorithm as the one described earlier. In other words, an algorithm such as *XrefU* can be used as a preprocessing tool to reduce the initial size of the enumeration tree in the BNB method. (5) It is still possible to further improve *XrefU* by "refreshing" the current list periodically and by superposing some localized optimization based on *greedy* techniques.

Discussed in the next section is an optimization technique to tackle the problem of allocating cross-referencing data units when the communication network has a finite flow capacity on each of its channel.

1.2. The capacitated model

When the flow in each communication channel of the network must be kept below a fixed level, called the flow capacity for this channel, the constraints of type (mip-1) and (mip-2) must be added to the mathematical model. The capacitated model is a mixed-integer (0-1) program where the objective function as well as the constraints are linear. More specifically, the model assumes the form of a minimum cost, multi-commodity flow problem with additional linear constraints involving the 0-1 variables. The algorithm presented in the coming section is based on the decomposition* technique for mixed-integer programs [MURT 76].

1.2.1. Decomposition of the capacitated model

The capacitated model, with fixed 0-1 variables, and its dual, are written in a convenient form for future calculations.

The mixed-integer program (MIP) was first presented in Section 5.2. *When all the integer* (0-1) *variables are fixed,* (MIP) assumes the form of a standard linear program (LP). For clarity the model will be rewritten in the following form:

$$(LP): \quad minimize \sum_{(i,j)} c_{ij} \sum_{s \in N} x_s(i,j)$$

subject to:

$$-x_s(i,N) + x_s(N,i) = -ASM_{si} - RDC_{si} - UPD_{si} \qquad \text{(lp-1)}$$

$$0 \leq \sum_s x_s(i,j) \leq K_{ij} \qquad \text{(lp-2)}$$

The only difference between (LP) and (MIP) is that in (LP) the variables to deal with are the real flow variables. (LP) is indeed a capacitated minimum cost multi-commodity flow problem.

Let $x^{r(s,i)}$ or simply x^r denote the node-arc flow vector for the r^{th} commodity (associated with a source s and a sink i) for $r = 1$ to p $(p = |N| \times |N|)$.

$$\text{thus} \quad x_s(i',j') = \sum_{i \in N} x^{r(s,i)}(i',j'), \quad \text{for any } (i',j') \in A.$$

Introducing these new variables, (LP) can be rewritten using the node-arc (*na*) formulation:

*Also called projection or partitioning.

$$(NA\text{-}LP): \quad minimize \sum_{(i,j)} c_{ij} \sum_{r} x^r(i,j)$$

subject to:

(i) *Capacity constraints:*

$$x^1 + x^2 + \cdots + x^r + \cdots + x^p \leq K \qquad \text{(na-lp-1)}$$
$$r \overset{\Delta}{=} (s,i) \quad s,i \in N,$$

where $x^r \overset{\Delta}{=} (x_1^r, \cdots, x_{|A|}^r)^T$ and $K \overset{\Delta}{=} (K_1, \cdots, K_{|A|})^T$.

(K_j is the flow capacity of arc $j \in A$.)

(ii) *Flow conservation constraints:*

$$Ex^r - q_r V_r = 0 \qquad \text{(na-lp-2)}$$
$$r = 1 \; to \; p$$

$$x^r \geq 0 \qquad \text{(na-lp-3)}$$

where $V_r \overset{\Delta}{=} -[ASM_r + RDC_r + UPD_r]$ is the flow value of node-arc flow vector x^r, E is a $(|N| \times |A|)$ node-arc matrix, q_r is a $(|N| \times 1)$ node-arc vector corresponding to arc $r \overset{\Delta}{=} (s,i)$:

$$q_r \overset{\Delta}{=} (0,0, \cdots 0,\underset{s}{1},0, \cdots 0,\underset{i}{-1},0, \cdots 0)^T$$

The arc capacity constraints of (i) lead to the *master program* of (NA-LP) , and the flow conservation equations of (ii) lead to the *subprogram* of (NA-LP). The subprogram breaks up into p independent subproblems which do not have variables in common. For the r^{th} subproblem, the constraints are:

$$Ex^r - q_r V_r = 0 \quad x^r \geq 0$$

Every basic vector for (na-lp-2) consists of flow variables associated with arcs in a spanning tree in G. In the *basic feasible solution* corresponding to a feasible basic vector for (na-lp-2), the flow value on all the arcs in the simple chain from s to i in the associated spanning tree is equal to V_r (V_{si}) , and the flow on all the other arcs is zero. Thus the r^{th} subproblem can be interpreted as a shortest chain problem from s to i in G. The *arc-chain* formulation of (LP) (as opposed to the node-arc formulation above) makes use of the chains $D_1^r, D_2^r, \cdots, D_{d_r}^r$ from s to i, and decides how much to ship along each of these chains [MURT 81]. Following the notation of [MURT 81], let x_h^r represent the quantity of the r^{th} commodity shipped along the chain D_h^r ($h = 1, \cdots, d_r$ and $r = 1, \cdots, p$); the amount of flow of the r^{th} commodity on an arc (i', j') is:

$$x^{r(s,i)}(i', j') \overset{\Delta}{=} \sum_{\substack{h=1 \\ (i',j') \in D_h^r}}^{d_r} x_h^r$$

$$\text{and } V_r = \sum_{h=1}^{d_r} x_h^r .$$

The master program corresponding to the multicommodity minimum cost flow problem (NA-LP) stated in arc-chain formulation is:

$$(AC\text{-}MLP): \quad \textit{minimize } z = \sum_{r=1}^{p} \sum_{h=1}^{d_r} \sum_{\substack{u: \\ e_u \in D_h^r}} c_u x_h^r$$

subject to:

$$\sum_{r=1}^{p} \sum_{\substack{h: \\ e_u \in D_h^r}} x_h^r \le K_u \qquad \text{(ac-mlp-1)}$$

$$u = 1, \cdots, |A|$$

$$\sum_{h=1}^{d_r} x_h^r = V_r \qquad \text{(ac-mlp-2)}$$

$$r = 1, \cdots, p$$

$$x_h^r \ge 0 \qquad \text{(ac-mlp-3)}$$

$$r \overset{\Delta}{=} (s,i) \quad h = 1, \cdots, d_r$$

The number of constraints in the master program is $|A|+|N|^2$, and hence the basic vectors of the master program will consist of $|A|+|N|^2$ variables.

$$c^r = (g_1^r \cdots g_{d_r}^r) \quad (1 \times d_r) \text{ vector}$$

$$g_k^r = \sum_{\substack{u: \\ e_u \in D_k^r}} c_u$$

$$\bar{x}^r = (x_1^r, \cdots, x_{d_r}^r)^T \quad (d_r \times 1) \text{ vector}$$

$$B^r = \begin{bmatrix} a_{11}^r & . & a_{1d_r}^r \\ \vdots & \vdots & \vdots \\ a_{u1}^r & . & a_{ud_r}^r \\ \vdots & \vdots & \vdots \\ a_{|A|1}^r & . & a_{|A|d_r}^r \end{bmatrix} \quad (|A| \times d_r) \text{ matrix}$$

$$a_{uk}^r = \begin{cases} 1 & \text{if arc } u \text{ is on chain } D_k^r \\ 0 & \text{otherwise} \end{cases}$$

$$b = (K_1 \cdots K_u \cdots K_{|A|})^T \quad (|A| \times 1) \text{ vector}$$

$$d^r = (1\ 1\ 1 \cdots 1) \quad (1 \times d_r) \text{ vector}$$

$$V^r = ASM_r + RDC_r + UPD_r \quad (1 \times 1) \text{ scalar}$$

$$V = (V^1, \cdots, V^{|N|^2})^T \quad (|N|^2 \times 1) \text{ vector}$$

$$r = (s,i) \text{ for some } s \in N,\ i \in N$$

Table 6.7 Notations for the capacitated model.

After some further change in notation (Table 6.7), (AC-MLP) can be rewritten as:

$$(AC\text{-}MLP): \quad \text{minimize } z = c^1\bar{x}^1 + \cdots + c^r\bar{x}^r + \cdots + c^p\bar{x}^p$$

subject to:

$$-B^1\bar{x}^1 - \cdots - B^r\bar{x}^r - \cdots - B^p\bar{x}^p \geq -b \qquad \text{(ac-mlp-1)}$$

$$\begin{aligned} d^1\bar{x}^1 &= V_1 \\ &\vdots \\ d^p\bar{x}^p &= V_p \end{aligned} \qquad \text{(ac-mlp-2)}$$

$$\bar{x}^r \geq 0 \qquad \text{(ac-mlp-3)}$$

There are $|A|$ constraints of type (ac-mlp-1) and $|N|^2$ constraints of type (ac-mlp-2). Let π be a $(1\times|A|)$ dual vector corresponding to constraints (ac-mlp-1) and μ be a $(1\times|N|^2)$ dual vector corresponding to constraints (ac-mlp-2). The dual of (AC-MLP), which we will call (DL) is:

$$(DL): \quad maximize \ -\pi b + \mu V$$

subject to:

$$\begin{aligned} \pi B^1 + \mu^1 d^1 &\leq c^1 \\ &\vdots \\ \pi B^p + \mu^p d^p &\leq c^p \end{aligned} \qquad \text{(dl-1)}$$

$$\pi \geq 0 \ \ and \ \ \mu \ \ unrestricted. \qquad \text{(dl-2)}$$

(DL) has $|A|+|N|^2$ variables but has a very large number of constraints, namely $\sum_{r=1}^{p} d_r$ constraints.

In the next sections an algorithm (*XrefC*) that generates improved feasible solutions for the capacitated model is described and discussed.

1.2.1.1. Optimizing the capacitated model

A projection technique [MURT 76, SALK 75] is used to solve the capacitated model.

We have shown above that the capacitated model can be formulated as a *minimum cost, multi-commodity flow, arc-chain* problem (AC-MLP), when all the 0-1 variables are fixed. Its dual, denoted (DL), was stated. Furthermore, the domain

$$\Delta \overset{\Delta}{=} \{ (\pi,\mu) : (dl\text{-}1)\text{and} \ \ (dl\text{-}2) \ \}$$

is independent of the 0-1 variables and is convex.

Proposition 18: *The domain Δ is a convex polyhedron.*

proof: That Δ is a polyhedron is obvious since the constraints (dl-1) and (dl-2) are linear. Next, if the dual problem (DL) is feasible, and its objective function is unbounded above (on the dual feasible domain Δ), then the primal problem (AC-MLP) cannot have a feasible solution. (This follows from a corollary of the *weak duality theorem*). If Δ were to be concave, then any feasible dual solution (π,μ) would have an unbounded objective, which would mean that for any value of the 0-1 variable vector, the corresponding primal solution (the flow) is infeasible. But the flow is only infeasible in the particular situation where there is not enough capacity to accommodate it for a specific repartition of data. We assume that there is enough capacity in the network to satisfy the flow for at least one allocation of data, and therefore Δ must be convex. •

For simplicity it will be assumed further that Δ is bounded. In practice this means that for any value of the 0-1 variables $X_{\ell k i}^{p q}$, $X_{\ell k}^{p}$ and Y_i^h, there exists a feasible flow (although not necessarily the flow that would have been obtained when the capacity constraints are omitted). In the context of this thesis, this is a very reasonable assumption because the problem under study is the design and usage of a distributed database on an existing computer network.

The maximum of $-\pi b + \mu V$ over Δ occurs at an extreme point of Δ. We denote the extreme points by (π^t, μ^t) $(t = 1, \cdots T)$. Therefore problem (DL) may be written as:

$$(DL'): \quad \underset{t=1 \cdots T}{maximize} \; -\pi^t b + \mu^t V.$$

And thus (MIP) becomes a full integer problem (IP):

$$(IP): \quad \underset{X}{minimize} \left\{ \underset{t=1 \cdots T}{maximum} \; (-\pi^t b + \mu^t V) \right\}$$

subject to: $(mip\text{–}1)$, $(mip\text{–}2)$, $(mip\text{–}3)$ *and* $(mip\text{–}4)$.

Algorithm *XrefC* follows:

ALGORITHM XrefC:

```
NoXtremePts := 0
X := $\overline{X}$ /* Denotes a feasible 0-1 solution,
where X stands for ( $\cdots X_{\ell k i}^{pq} \cdots X_{\ell k}^{p} \cdots Y_i^h \cdots$ ) */
UpperBnd := $+\infty$
while ($z <$ UpperBnd $- \epsilon$) do
begin
    <$\pi,\mu$>[NoXtremePts+1] := Solve(DL)
    NoXtremePts := NoXtremePts + 1
    if UpperBnd >
    max $\{-\pi^t b+\mu^t V(X)\}_{(t=1,\ \cdots,NoXtremePts)}$
    then UpperBnd := max $\{-\pi^t b+\mu^t V(X)\}$
    X := Solve(IP)
end
Flow := Solve(AC–MLP).
/* Solve for $x_h^r$ for all $r = 1$ to $p$
and all $h = 1$ to $d_r$ . */
```

end XrefC.

To generate the extreme points of Δ , the arc-chain formulation of the primal program (AC-MLP) is solved rather than the direct program (DL). A column generation version of the revised simplex method with the product form of the inverse tableau [MURT 76, MURT 81] is used. It is known that the computational effort involved tends to be smaller, and the numerical stability greater than would be the case in carrying out the same algorithm using the explicit form of inverses. Furthermore the pivot matrices occupy very little storage space. The dual solution (π,μ) is read directly from the inverse tableau. Therefore (AC-MLP) can be solved optimally and very efficiently to find the dual solution (π,μ) and the flow vector $x^{r(s,t)}$. The main obstacle to finding an optimal solution for the capacitated model comes from the difficulty of solving (IP) optimally in *XrefC* as was the case when solving the *uncapacitated* model with BNB. For the *uncapacitated* model, an adaptive random search algorithm (*XrefU*) was described that can handle the constraints on the 0-1 variables, and can generate good feasible solutions in a relatively reasonable amount of CPU time. Of course it cannot be guaranteed that *XrefU* will generate an optimal solution for the *uncapacitated* problem, but it may be argued that looking for the true optimal solution is a luxury not worth paying for anyway. In the context of *XrefC*, the problem of solving a large integer program (IP) still remains, and the questions that must be addressed are the following. (1) In each main iteration of algorithm *XrefC*, is it acceptable to use suboptimal values for the 0-1 variables X and still preserve the meaningfulness of the underlying projection technique? (2) If the answer of (1) is yes, and if an adaptive random search is used to solve (IP), how robust will *XrefC* be? In the next section question (1) will be answered and the issue raised by question (2) will be discussed.

1.2.1.2. Convergence and robustness of XrefC

It is argued that Algorithm *XrefC* features the robustness of an adaptive search.

Problem (DL) has a finite number of solutions (π,μ). However, if (IP) is not solved optimally within each iteration of *XrefC*, then a dual solution (π,μ) may be produced more than once. If (IP) is allowed to terminate whenever $z^{k+1} \leq z^k - \epsilon$, for some positive ϵ, then the incumbent objective z must be improved at each repetition of a dual solution. But since the optimal value of (MIP) is bounded below, a dual solution can only be repeated for a finite number of times. Therefore if the algorithm used to solve (IP) eventually converges, then *XrefC* must converge too. If (IP) is solved with an adaptive random search, as suggested in the previous section, nothing can be said about the convergence of *XrefC*, because it is known that the adaptive search does not necessarily converge [HOLL 75]. However, each iteration of the adaptive algorithm used to solve (IP) will generate improved solutions for (MIP). Finally, it is interesting to remark that the convergence of the algorithm used to solve (IP) is a sufficient condition for the convergence of *XrefC* but it is not necessary. Therefore the answer to question (1) is: *XrefC* is still meaningful even though (IP) is not solved optimally during each iteration.

Before discussing question (2) it is helpful to recall what is meant by the notion of *robustness* in the context of a random search algorithm; as was mentioned in Section 2, a random search is said to be *robust* if the number of trials allocated to the observed best solutions increases exponentially with respect to the remainders in an observed sample. An adaptive search such as *XrefU* exhibits such a property. What can be concluded about *XrefC* when an adaptive search is used to solve (IP)? In the following paragraph the notation shown in Table 6.8 will be used.

Consider Figure 6.4 which helps to visualize the execution of *XrefC* with respect to a discrete variable t_j $(j \geq 0)$ corresponding to the iteration process in the adaptive algorithm used on (IP). On the axis of Figure 6.4, each interval $[t_j-1, t_j)$ corresponds to one execution of

---

Structure-schema: The set of all stuctures that display a *predefined pattern of 0's and 1's at specific positions.*

e.g.

$\boxed{0\,1\,0\,1\,1\,0\ldots 0\,0\,1\,0}$

is an instance of structure-schema:

$\boxed{0\,1\,\lambda\,1\,1\,\lambda\ldots\lambda\,0\,1\,0}$

For each structure-schema ξ:

$M(t)$	Number of instances of ξ in the list of structures, at step t.
$N_j(t)$	Number of trials allocated, from t_j to t, to structures which are instances of ξ.
$n_j(t)$	Number of trials allocated, from t_j to t, to structures which are not instances of ξ.
t_0	Initial step.

Table 6.8. Additional definitions for the adaptive algorithm.

the main "do while" loop of *XrefC*, except for the line $X := Solve(IP)$, whereas an interval $[t_{j-1}, t_{j+1}-1)$ corresponds to one execution of line $X := Solve(IP)$. Given a structure-schema ξ, the following was proven in [HOLL 75]:

$$N_j(t) \geq M(t_j)e^{Z_j n_j(t)} \quad t_j \leq t \leq t_{j+1}-1.$$

In this context, for any t the following can be written:

$$N_{t_0}(t) = \sum_{\substack{j=0: \\ t_l < t}}^{l-1} N_j(t_{j+1}-1) + N_l(t)$$

$$\geq \sum_{\substack{j=0: \\ t_l < t}}^{l-1} M(t_j)e^{Z_j n_j(t_{j+1}-1)} + N_l(t).$$

In the course of algorithm *XrefC*, a new constraint is added to (IP) whenever a new dual solution is generated, therefore:

$$M(t_j) = M(t_j-1) - r(j)M(t_j-1)$$

where $0 \leq r(j) \leq 1$ and $\lim_{j\to\infty} r(j) = 0$. A lower bound for $N_{t_0}(t)$ can now be rewritten as:

$$N_{t_0}(t) \geq \overline{N_{t_0}}(t) - \sum_{\substack{j=0: \\ t_l < t}}^{l-1} m(t_j)e^{Z_j n_j(t_{j+1}-1)} + N_l(t)$$

where $\overline{N_{t_0}}(t) \overset{\Delta}{=} M(t_0)e^{Z_0 n_0(t_{l-1}-1)}$,

and $m(t_j) \overset{\Delta}{=} M(t_j) - M(t_j-1)$.

Both an increase in j as well as an increase in t contribute in making $m(t_j)$ decrease, so that eventually:

$$\overline{N_{t_0}}(t) >> \sum_{j=0}^{l-1} m(t_j)e^{Z_j n_j(t_{j+1}-1)},$$

in which case $N_{t_0}(t) \geq M(t_0)e^{Z_0 n_0(t)}$ which indicates robustness. Informally, *XrefC* is said to exhibit the same kind of robustness as does *XrefU* "in the course" of its execution.

In the next chapter some practical remarks are drawn from the usage of the CRDU-distribution model.

Old Structure A $\boxed{0\,1\,0\,1\,1}$ $\boxed{0\,1\,1\,1}$

Old Structure B $\boxed{1\,1\,1\,1\,0}$ $\boxed{0\,0\,0\,1}$

Yield:

New Structure A $\boxed{0\,1\,0\,1\,1\,0\,0\,0\,1}$

New Structure B $\boxed{1\,1\,1\,1\,0\,0\,1\,1\,1}$

[i] Cross-over at position 5

Old Structure $\boxed{0\,1\,0\,1}$ $\boxed{0\,1\,1}$ $\boxed{1\,0}$

Yields:

New Structure $\boxed{0\,1\,0\,1\,1\,1\,0\,1\,0}$

[ii] Inversion between positions 5 and 7

Old Structure $\boxed{0\,1\,1\,0\,0\,1\,1\,0\,1}$

Yields:

New Structure $\boxed{0\,1\,1\,0\,1\,1\,1\,1\,1}$

[iii] Mutation at positions 5 and 8

Figure 6.4. **The string manipulation operations used in an adaptive search**

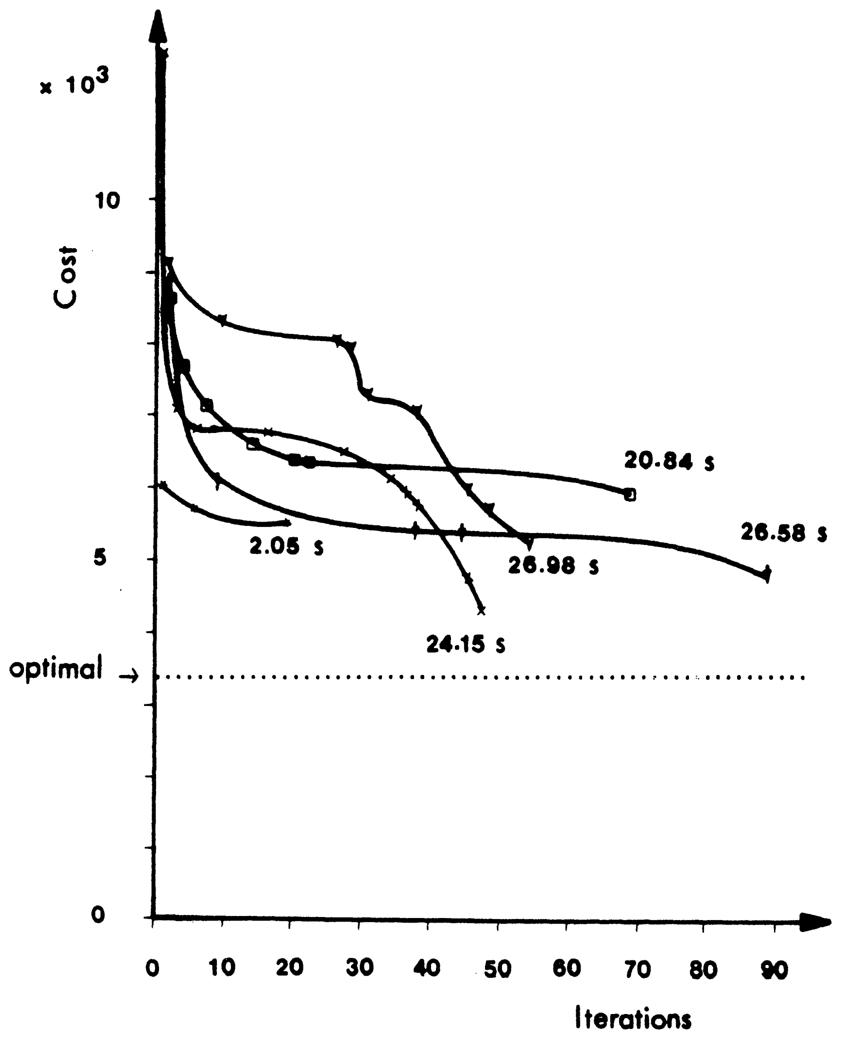


Figure 6.5. The performance of Algorithm *XrefU*

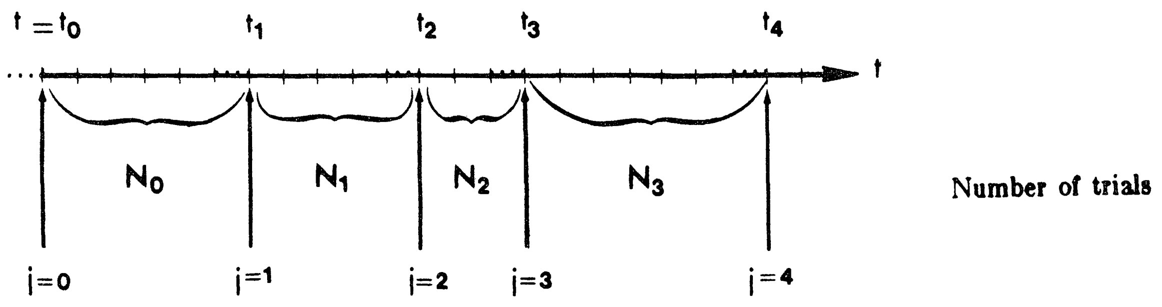


Figure 6.6. A schematic visualization of the execution of Algorithm *XrefC*.

CHAPTER 7

EXPERIMENTAL RESULTS

Some practical observations are made concerning the distribution of CRDUs in a computer network. The effect of the reducing and assembly rates*, over the simultaneous referencing of CRDUs is investigated (Section 7.1). Also investigated, the effect of the updating rates over cross-referencing (Section 7.2).

7.1. Effect of the reducing rates over cross-referencing

A first experiment consists of investigating the effect of the reducing rates, in the matrix [$A_{\ell}^{p}{}_{k}^{qr}$], over the cross-referencing aspect in distributed query processing. The results of such an experiment can be of practical interest for the designer of a distributed database. Indeed, cross-referencing is acknowledged to be one cause for the need of restructuring standard file-allocation models. (See Chapter 2. The other cause being the clustering effect). Hence, if it is observed that, beyond a certain threshold, the reducing rates become too large compared with the assembly rates, then it can be concluded that cross-referencing is not a viable alternative to CRDU duplication. In practice, this implies that any query envelope is likely to be clustered at a single site rather than partitioned and distributed over several sites. In this case, the CRDU-distribution problem becomes quite similar to a file-allocation problem, where each query envelope is a file. On the other hand, if it is observed that the reducing rates are not likely to impair the performance of transaction processing in the computer network, then, data redundancy is not likely to supersede to the practice of cross-referencing.

*In this chapter, the term "rate" will generally be favored to the term "flow", since the former has a more numerical connotation than the latter.

In the latter case, file-allocation techniques may not be applied since they do not satisfactorily capture the transaction processing environment of the distributed database.

Ten input data files, corresponding respectively to Problems 1-10, were set up for Algorithm *XrefU*. Each input-file specified the MRDB problem as before (Chapter 6 and Appendix D), but the entries of the type $A_{\ell}{}^{p}{}_{k}^{qr}$, and only those, were incrementally decreased until they were set equal to zero in the tenth input-file. All other parameters were kept constant, and in particular the updating rates were fixed.

Some new definitions are made. It is assumed that the input-file of a problem is given in the form of tables as shown in Appendix D. All the definitions below apply to the input-file of a problem, and to the values of the variables appearing in a feasible solution of that problem.

The measure of cross-referencing, ξ, is defined to be:

$$\xi \overset{\Delta}{=} \frac{\begin{array}{c}\textit{Number of non-zero variables corresponding to clusters}\\ \textit{in the same partition but different sites}\end{array}}{\textit{Total number of non-zero materialization variables}} .$$

The measure of redundancy, ρ, is defined to be:

$$\rho \overset{\Delta}{=} \frac{\textit{Number of non-zero allocation variables}}{\textit{Total number of allocation variables}} .$$

In the following notations, the subscripts t, and s, which normally denote a user-site, are omitted. For example, the symbol $A_{\ell}{}^{p}{}_{k}^{q}$ will be written as A_{ℓ}^{pq}. In this context, we are interested in the volumes of data associated with query processing sequences (entries in tables 3 and 5, Appendix D), not in the rates. The frequencies of occurence of queries are not considered. The total transaction volume, τ, is defined to be:

$$\tau \overset{\Delta}{=} \sum_{\substack{k \\ p \in PAR_k \\ q \in CLU_k^p}} A_k^{pq} + A_k^{pqr} + \bar{A}_k^{pqr} + \sum_{\substack{k \\ h \in WRT_k}} B_k^h .$$

The assembly volume, α, is defined to be:

$$\alpha \overset{\Delta}{=} \frac{\left[\sum_{\substack{k \\ p \in PAR_k \\ q \in CLU_k^p}} A_k^{pq} \right]}{\tau} .$$

The reducing volume, β, is defined to be:

$$\beta \overset{\Delta}{=} \frac{\left[\sum_{\substack{k \\ p \in PAR_k \\ q \in CLU_k^p}} \left[A_k^{pqr} + \bar{A}_k^{pqr} \right] \right]}{\tau} .$$

The reducing volume ratio, η, is defined to be:

$$\eta \overset{\Delta}{=} \frac{\beta}{\alpha} .$$

Figure 7.1 displays the results of the experiment. The measure of redundancy, ρ, is plotted with respect to the reducing volume ratio, η. Each one of the ten problems was run with *XrefU*. The number of iterations varied between 9 and 19, and the CPU time varied respectively between 27.83 *seconds* and 60.27 *seconds* on the Honeywell 870. For each problem, *XrefU* was initiated with the same list of 20 structures. In Figure 7.1, we observe that redundancy tends to decrease with decreasing reducing volumes. This seems to confirm initial expectations, namely, that decreasing the penalty associated with cross-referencing, reduces the need for clustering query envelopes, and thus reduces the need for CRDU duplication. Further, the cross-referencing is found to start at 22 % for Problem 10, and to plateau at 10 % for the first problems. Note that, when the reducing volume ratio becomes quite large, we expect the cross-referencing to be quite small, and therefore, CRDU-distribution can be solved with conventional file-allocation techniques.

The next experiment we conducted, was to investigate the effect of the updating volume, to be defined, over cross-referencing.

7.2. Effect of updating over cross-referencing

A second experiment consists in investigating the effect of the updating volume over cross-referencing.

As in Section 7.1, a measure of cross-referencing is provided by ξ. The measure of updating volume, μ, is defined as:

$$\mu \overset{\Delta}{=} \frac{\left[\sum_{\substack{k \\ h \in WRT_k}} B_k^h \right]}{T} .$$

As before, the input-files of the test problems were set up for Algorithm *XrefU*. All input-files had identical entries for the MRDB problem, but for the entries B_k^h, and only those, which were increased incrementally. Figure 7.2 displays the results of this experiment. It shows that for small values of μ, redundancy, as measured by ρ, increases. This is obviously expected since, when updates are cheaper, it is practical to duplicate CRDUs at different sites, and thus, the need for cross-referencing becomes insignificant. Indeed, whenever the updating volume is small, it is probably quite appropriate to make use of file-allocation techniques rather than CRDU-distribution techniques to solve the distribution problem. For high values of μ, duplications of CRDU copies becomes more expensive, and thus, ρ decreases. On the other hand, cross-referencing becomes a practical aspect of distributed query processing since the cost associated with the reducing flow can compete with the cost associated with the updating flow. In this latter case, where updating rates are significant, file-allocation techniques are certainly inappropriate to solve the distribution problem.

7.3. Lower and upper bounds for the communication cost

If all the constraints of the *uncapacitated* model are eliminated, then the lowest possible cost is obviously equal to zero. For the unconstrained model the upper bound for the cost is simply equal to τ (page 187). For the problem solved earlier in Section 6.1.2 (page 159) this upper bound τ is equal to 187,920 *kilobits/hour*. However, the communication cost is never allowed to attain the value of τ, since τ corresponds to an infeasible solution for the CRDU distribution model. In practice, setting all the zero-one variables to one leads to a possible but highly unreasonable solution. Indeed, every query is solved through the simultaneous accessing of all the possible partitions of its pre-assigned query envelope. Further, since all the allocation variables are equal to one, there is usually more CRDU duplication than is needed to solve most queries.

With the constraints, the lower bound for the cost is of course the optimal solution sought. The upper bound for the cost is, a priori, as hard to find as the optimal cost. However, we observe that a feasible solution with largest cost can be obtained as follows: (1) Set all the allocation variables to one. (2) For every query Q_{tk}, select a partition p of the pre-assigned set ACC_{tk} which maximizes $\sum_{q,r \in CLU_{tk}^{p}} A_{t}{}_{k}^{p}{}^{q} + A_{t}{}_{k}^{p}{}^{qr} + \overline{A}_{t}{}_{k}^{p}{}^{qr}$. (3) Whenever possible, for every set ACC_{tk}, and for a chosen partition p of ACC_{tk}, choose the materialization variables $X_{t}{}_{k}^{p}{}_{i}^{q}$, $X_{t}{}_{k}^{p}{}_{j}^{r}$ $\cdots$ so as to place cluster q at site i different from t, cluster r at site j different from both i and t, and so on... For most non-trivial models, i.e. when the computer network contains at least three sites, finding the upper value of the cost is simple. For the problem solved in Section 6.2.2, this upper bound is found to be 94,960 *kilobits/hour*. Note that the best feasible solution found by Algorithm *XrefU* (4,360 *kilobits/hour*, page 171) has an error equal to $((94{,}960 - 3{,}390)/(4{,}360 - 3{,}390)) \times 100 = 1.06\%$ when compared with the optimal solution (page 159).

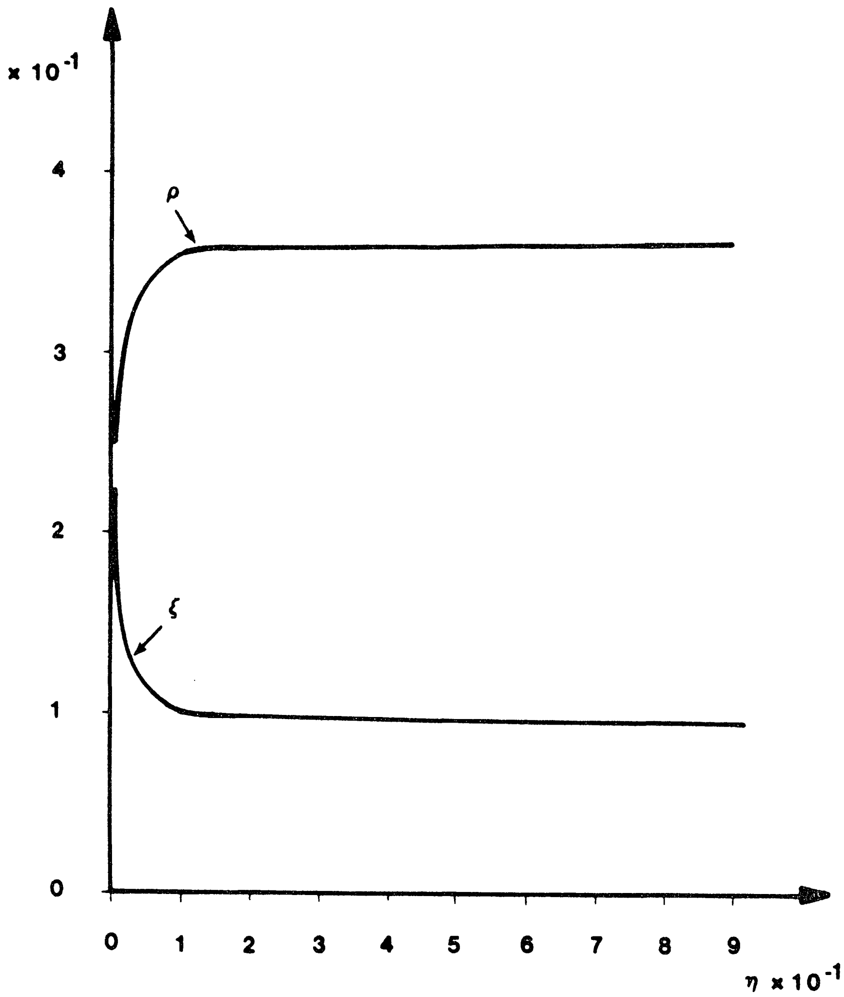


Figure 7.1. Variations of the measure of redundancy ρ and the measure of cross-referencing ξ with respect to the reducing volume ratio η

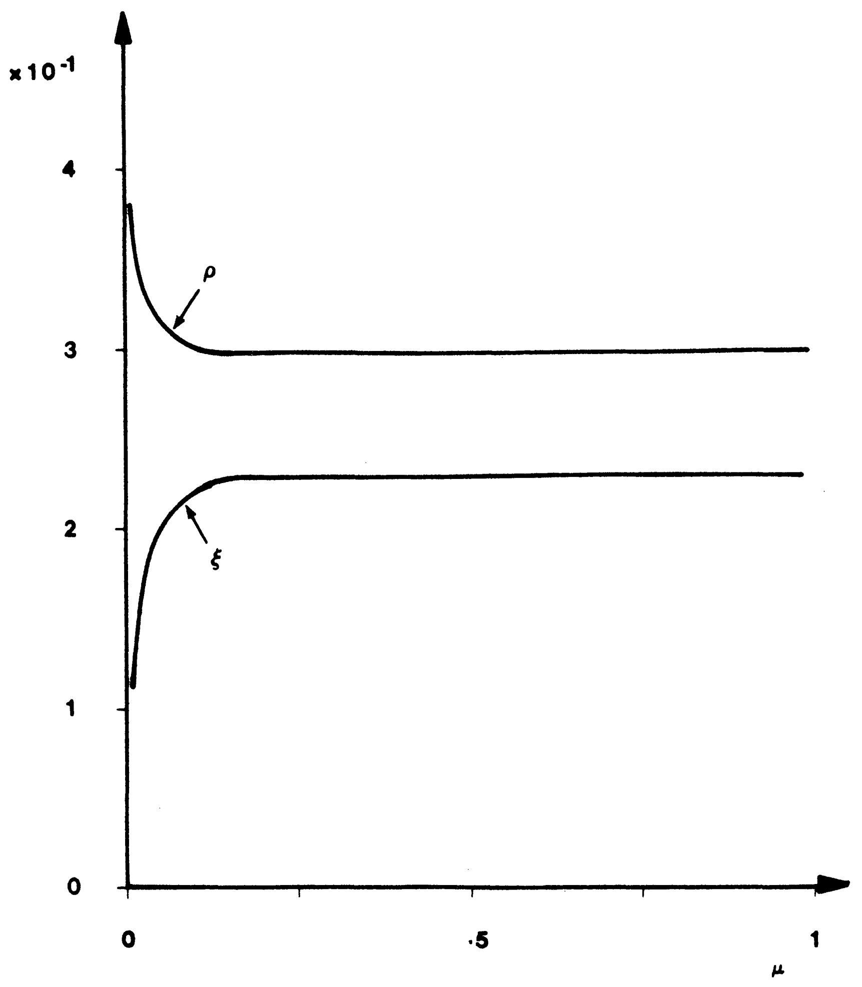


Figure 7.2. Variations of the measure of redundancy ρ and the measure of cross-referencing ξ with respect to the updating volume ratio

CHAPTER 8

CONCLUSION

This research addresses the distributed database design problem. In this context, this problem consists of *selecting* and *distributing* the building blocks of a database which will be shared by the nodes of a computer network. No a priori data-files or relational tables are assumed to be provided by the user. Section 8.1 restates the problem of interest and reviews the objectives and accomplishments of the research. Section 8.2 discusses unsolved issues. Section 8.3 outlines the main contributions of this thesis. Finally, Section 8.4 suggests some possible improvements and issues which can be the subject of further investigations.

8.1. The problem

A summary of the problem follows. User provided inputs include:

(1) A *universal relation scheme* which consists of (i) a universe of data-items or *attributes,* (ii) a set of semantic constraints, of the *functional dependency* (FD) type, holding on the attributes, and (iii) a set of statistical informations for each attribute, the domain from which the attribute gets its values, and for each FD.

(2) A computer network with tightly or loosely connected nodes. It is assumed that the communication network has a fixed topology and that there is enough over-all channel capacity. Communication costs, and channel capacities are assumed to be given.

(3) The user's scheduled queries and updates. A query is assumed to consist of a *target list* of attributes, a frequency of usage, and a set of *qualification clauses* holding on attri-

butes. An update is assumed to consist of a target list of attributes, a *type,* i.e. whether the update is a "delete" or an "add", and a frequency of occurrence.

The first phase of the design process, the *selection,* consists in identifying the data-units which can serve as atomic building blocks for a distributed database, and which allow efficient distributed query and update processing. The data-units are to be involved in *cross-referencing* operations in the context of query processing sessions. They may be visualized as relational tables or fragments thereof, and are referred to as *cross-referencing data-units* (CRDUs). The CRDUs are mapped to functional dependencies holding on attributes. The selection part of the design process involves the consideration of semantic issues, such as the correctness of query strategies, and the consistency preservation of the database under updating. Since most logical accesses translate into data-volume flows due to cross-referencing and assembling operations, optimization assumes a critical role in selecting CRDUs. The modeling of semantic constraints using graphs and *matroids* results in efficient optimization algorithms. At the end of this design phase, the CRDUs which can be subject to distribution are well defined.

The second phase of the design process, the *distribution* , concentrates on the optimization of the operational communication cost. Typical issues addressed in this stage are: (i) The modeling of the cross-references of physically distant CRDUs. These cross-references are at the origin of *file-to-file* inter-site accesses which add up to the usual *file-to-user* accesses. Typically, modern strategies of distributed query processing cannot be captured by classical *file-allocation* models, and the inclusion of the "cross-referencing" ingredient in data-allocation, changes substantially the structure of mathematical programming models. (ii) The *clustering* of the CRDUs, e.g. whether they belong to a common relational table or not, and the resulting impact on the file-to-file and file-to-user flows of data-volume. (iii) The *materialization* , in the static sense, of the CRDUs, i.e. the choice of relevant instances of CRDUs to seek access to, for each query session, when there are duplicates. (iv) The accessing routes in the communication network, especially when flow capacity constraints must be accounted with. The distribution of the CRDUs is modeled in the form of a manageable *multi-commodity mixed-integer* program, with quadratic cost and linear constraints. *Branch-and-bound, adaptive* random search and *projection* techniques are used to solve the model.

In addition to providing a complete procedure for distributed database design, this research draws some practical conclusions as to the distribution of CRDUs in a computer network. For instance, the effect of the file-to-file data-volume with respect to the file-to-user data-volume over all queries, on the simultaneous-referencing of CRDUs is experimentally studied. Also, the effect of the flow-capacity constraints over optimal distributions and materializations of CRDUs, and over optimal access routes is illustrated.

The consideration of semantic issues gives a new dimension to the classical data-allocation problem. Indeed, the dependence of CRDUs, in this context, FDs, becomes explicit through both query and update processing strategies. Thus, a CRDU-allocation model is shown to necessitate more modeling power than classical file-allocation models.

8.2. Unsolved Issues

Some issues, to be mentioned below, have been left unsolved because we believe that existing techniques to solve them are appropriate, and therefore, expanding upon those issues would not bring about an original research contribution. However, the designer of a distributed database who wishes to implement the proposed design methodology, must be aware of those issues and must address them fully.

Of importance to the designer of a distributed database is the following question: What to do once the CRDUs, i.e. the FDs, have been distributed? Firstly, the designer must be aware that after Step-5 of the design procedure, i.e. after the distribution of FDs, some redundant FDs may have been allocated at any one site. Therefore, a nonredundant subset of FDs must be identified at each site, and not just any nonredundant cover. This problem has been addressed in the literature, e.g. refer to [KAMB 78]. Secondly, the synthesis of relational tables in third normal form, or better, may be accomplished using Kambayashi's algorithm [KAMB 78]. If, however, the local databases are to be physically implemented in the form of network databases, then, techniques such as Housel and Yao's [HOUS 79] may be considered. The implementation of distributed query strategies, using the principles of semi-joins for data reduction, will then require the addition of network-to-relational interfaces at each node of the computer network.

The problems mentioned above require some exercises of implementation. The contributions of the research are now reviewed.

8.3. Research contributions

The main research contributions are now discussed.

In Chapter 4, a new algorithm, *Reduce*, is proposed to derive an LR-minimum cover for a set of FDs. The algorithm is shown to be at least as good as Maier's algorithm [MAIE 80]. Further, it is argued that because Algorithm *Reduce* uses exclusively a graph structure during its execution, it can be more readily implemented, and is more direct in its approach, than Maier's algorithm.

A major contribution of this research, however, is the new dimension added to data-allocation. This is so because the database distribution problem involves the distribution of data-dependent CRDUs, as opposed to data-independent files. Data-dependence is at the origin of cross-referencing operations in distributed query processing, and of cascade operations in update processing. Further, the problem of data distribution to enhance system performance, is enlarged with the so-called envelope optimization problem for scheduled queries, and the problem of update cascades determination, in the first phase of the methodology (Chapter 4). The envelope optimization problem is modeled with simple graph theoretical tools which lend themselves to efficient graph processing algorithms. Similarly, the update cascades determination problem is tackled through direct graph processing techniques. The whole design methodology can thus be implemented as an automatic design package*.

In the second phase of the methodology (Chapter 5), in addition to the novelty of addressing the problem of distributing data-dependent units, i.e. CRDUs mapped to FDs, we managed to model the CRDU-distribution model in a compact form while adding important features previously non existing in file-allocation models. Indeed, the distribution model includes the modeling of (i) the flow associated with cross-references, and (ii) the effect, on the data-volume rates, of clustering CRDUs at the same site; (i) is accomplished by setting up the optimization program as a multi-commodity flow problem, and (ii) is handled by introducing zero-one partition variables. Note that without those latter variables, the distribution model would be much more complex, i.e. highly nonlinear in terms of the allocation variables, and therefore much harder to solve. The mathematical programming model assumes the form of a quadratic program, i.e. the cost has a quadratic structure and all the constraints are linear. This latter feature allows the usage of a wide variety of computational techniques. Some of those techniques are discussed in Chapter 6. A problem of reasonable size is solved optimally. Heuristic techniques are also discussed and implemented. Chapter 6 reports several positive numerical results. Chapter 7 indicates how several useful experiments can be performed, and suggests, through numerical experiments, that simplifying design decisions can be made by looking carefully at the user-provided inputs.

In the next section, areas of need for further research are pointed out.

8.4. Improvements and further research

The proposed methodology can be improved if some issues are dealt with more thoroughly. These issues will be discussed in the following paragraphs.

*With some reserves for updates. Refer to Section 4.5.

One important notion in database theory is the notion of user views. In this thesis, the semantic framework consists only of functional dependencies and join dependencies. Higher level semantic notions may be introduced to capture wider updating features, such as the independence of views with respect to updating [FEDA 80]. Those semantic notions may also seek to control update cascades to simple and predictable ranges. In general, such semantic considerations have as an effect to restrict the domain of possible CRDU distribution solutions.

The notion of universal relation scheme is found to be somewhat controversial by some researchers (e.g. see [AZKE 83]). However, this assumption is not central to the proposed design methodology, and can be relaxed. The notions of query, link and envelope would have to be specified with respect to a set of user-views and not only one universal relation scheme.

The concept of relation synthesis from FDs [BERN 76] is implicit in the proposed design methodology. Non functional relationships can usually be handled by FD modeling techniques [BERN 76], and are compatible with synthesis algorithms [BERN 76, KAMB 78]. However, the issue of handling non functional relationships needs to be captured in a more formal framework than the one available in [BERN 76].

Another critical notion for database distribution, is the notion of distributed query processing strategy. It has been shown that some types of database schemata are highly desirable to simplify distributed query processing sequences [BEER 81]. Thus, some further restrictions may be imposed on a distributed database and therefore feasible CRDU-distribution strategies may be restricted accordingly.

It was pointed out that distributed query processing strategies are very related to database distribution. Strategies based on static materializations have been assumed. This means that the processing of a query starts after the access of a non-redundant portion of the database, i.e. a non-redundant query envelope. However, if parallelism is taken into advantage of in query processing [WONG 81], other distribution configurations may be sought as a result.

To optimize, other more efficient heuristics must be investigated. Indeed, if a network contains hundreds of nodes, a large number of user-queries and CRDUs, then one may reasonably question the usefulness of the algorithms proposed for CRDU distribution. We claim that the distribution model itself is practical, even for large applications, but solution techniques may be improved to reduce storage requirements and to keep CPU time under reasonable bounds. Larger and larger problems can now be solved optimally using standard optimization techniques, and it can be expected that problems with tens of thousands or even hundreds of thousands variables will be solvable in reasonable CPU times. However, efficient heuristic techniques, parallel algorithms and distributed algorithms may be considered to be possible present alternatives for enhanced computing performance.

Finally, more thorough testing should be accomplished. In Chapter 7, the results, although plausible and reasonable, are dampened by the fact that the distribution algorithms are quite expensive to execute. We were therefore content to obtain good solutions for each problem that was tackled, but the experiment would have gained to be based on closer-to-optimal solutions, and on a higher sample of solutions. Also, the notions of cross-referencing and redundancy are very input-data-dependent. Therefore, very careful experiments should be based on the control of such parameters as, the noncriticality of cross-references, the average size of query envelopes etc. In general, the dependence of various parameters should be explored more thoroughly. In the same spirit, measures of performance as well as measures of system characteristics should be studied in a formal way using stochastic modeling and statistical analysis.

APPENDICES

APPENDIX A

INDEX

Antecedent: The left-hand-side of a functional relation from a set of data values to another set of data values, i.e. the "domain" of the function (in the usual sense of the term).

Assembly flow: The files to be read during the querying of the database are processed locally and reduced in an effort to eliminate unnecessary information. When this is accomplished the reduced files are sent to a common node in the network and assembled at that node to produce a final answer. The flow produced by this latter process is called assembly flow.

Attribute: In APPENDIX B, PATIENT and WARD are two different attributes (also called "data-items").

Autonomous: Autonomous computers work in an "equal partnership" relationship. No computer receives orders from another one.

Availability: The availability of a file is the probability that at any instant of time, at least one copy of that file in the network resides at a site which is accessible from all other sites in the network [KHAB 80].

Centralized computing system: A computing system that operates under the sole supervision and control of a single selected computing center.

Centralized database: A complete database, located in one host computer, which features a minimum amount of data duplication.

Computer network: A network that has computers for nodes, and communication channels for edges. For large networks, the channels are usually telephone lines or satellite links.

Conceptual schema: The overall data model which captures, in a formal way, the information relevant to an organization.

Data-unit: A non dissociable piece of data which will be copied at one or several sites of a computer network.

Data-item: The atomic data object. In this work, "data-item" and "attribute" are synonymous.

Decomposition: The application of formal rules on a data file to derive file fragments which can be recombined to recover the original file. Data integrity is preserved.

Deletion cascade: A sequence of deletions of data, triggered by a single deletion. Deletion cascades preserve database consistency.

Distributed database: The geographical distribution of a database among the sites of a computer network. An important feature of distributed databases is data duplication for increased data availability and parallelism.

Distributed computing system: A computing system based on the distribution of processing and control functions over the sites of a computer network.

Distributed query processing: The concurrent access of distant files in a computer network to provide an answer to a query. Distributed query processing leads to the transmission of data between some of the files accessed in order to eliminate nonrelevant information. The reduced files are then moved to one single site to be assembled.

Domain: In APPENDIX B, CHARACTER(15), is the domain of attributes NAME and PHYSICIAN. The domain of an attribute is the value set from which this attribute can pick its values.

Envelope: A relevant portion of the database that is sufficent to solve a query, and that contains no extraneous information to provide a correct answer to that query.

External schema: A subset of the conceptual schema that is relevant to one user of an organization.

Fragment: A file reduced through local processing at a computer site.

Functional dependency: Semantic constraints holding on some attributes. FDs are used to identify data logical accesses and constitute a basis for data integrity.

Horizontal network: Network of autonomous, logically equal, computers (see Autonomous).

Integrity: The ability to disclose correct information when correct reading and querying methods are applied.

Inter-file flow: The files to be accessed in a querying of the database are processed locally and reduced by each other, whenever possible or advantageous, in an effort to eliminate unnecessary information. This is accomplished by the transmission of common data between each pair of files.

Join: A formal operation aiming at merging relations which possess common data. In APPENDIX B, relation R12 is the join of relations R1" and R2' over the domain of NAME. We write R12 = R1"[NAME]R2'.

Logical database schema: A set of records (relations), each one identified by a heading or list of data-items (attributes).

Logical design: The definition of clusters of data-items (attributes) to determine records (relations), taking into consideration the cost, measured in data volume, of accessing a set of data-items for every query.

Lossy: That causes the creation of misleading information due to the omission of relevant initial information.

LR-minimum cover: A set of FDs F is LR-minimum, if:

(1) There is no set G with fewer FDs than F such that $G^+ = F^+$ (section 3.1.1),

(2) for every FD $X \rightarrow Y$ in F, there is no $\overline{X}$ properly contained in X with $\overline{X} \rightarrow Y$ in F^+,

(3) replacing FD $X \rightarrow Y$ in F by $X \rightarrow \overline{Y}$, with $\overline{Y}$ properly contained in Y, alters the closure of F.

Materialization: A representative of data sought by a distributed query and chosen to take advantage of data-redundancy and parallelism. In this work, a nonredundant representative of data to be accessed for the solving of one query.

Normalization: Applied to relational tables in a relational database: It is the restructuring, if required, of relations to force certain rules between attributes to be satisfied. The normal form of a relation is defined with respect to the functional dependencies holding on the attributes representing this relation (see Functional Dependency, Attribute, Relation).

Projection: In APPENDIX B, relation R1' is a projection of relation R1 over attributes NAME,WARD. We write R1'=R1[NAME,WARD].

Query: A question to the database. In the general sense queries are represented by a set of referenced attributes or target list.

Query optimization: The process of choosing a set of data and a strategy to process the data so as to solve a query and minimize its cost.

Record: It is identified by a list of data-item names (see Relation).

Reducing operation: Query processing operation aiming at eliminating data that are of no interest for a query result. Each reducing operation can be mapped to Projection, Selection or Join (see Inter-file flow, Projection, Selection, Join, Semi-join).

Redundancy: One data element can have many duplicates in the actual database. In particular, in a distributed database, data may be duplicated at different sites.

Relation: A table of rows and columns. Each column is identified by an attribute name. Each row, also called a tuple, contains one value from the domain of each column-attribute (APPENDIX B; see Record, Logical database schema).

Relational algebra: A set of operations with which relations may be manipulated to achieve any desired tabular representation, i.e. to "cut" them or to combine them whenever possible. The three main operations are Selection, Projection, and Join.

Relational model: Two dimensional tabular representation of data. APPENDIX B shows relations R1, R2 and R3 representing a database in the relational model. The relational model was defined to enhance easy understanding by people with little or no training in programming (nonprocedural), evolution (additions and deletions) without major logical restructuring, flexibility of use of data (see Attribute, Domain, Relation, Tuple, Relational algebra).

Reliability: The network reliability is the probability that at any instant of time the network is connected

Selection: In APPENDIX B, relation R2' is a restriction of relation R2 based upon the clause TEST=GLUCOSE.

Semantic: Relating to the meaning of data.

Semi-join: The semi-join of relation R by relation S on attributes X, denoted by R<X]S is defined as $(R[X]S)[Att_R]$ (see Projection, Join), where Att_R denotes the attributes of R. If R and S are stored at different sites, R<X]S can be obtained by moving S[X] to the site that holds R. In APPENDIX B, relation R12' is the semijoin of relation R1" by relation R2'.

Site: A node of a computer network. A site corresponds to a computing element, also called a host computer.

Storage redundancy: The duplication of identical data at one site is not desirable if it leads to extraneous information at that site. Storage redundancy at one site indicates that data can be eliminated at that site without any loss in information, thus reducing storage cost. Otherwise, storage redundancy in the network (see Redundancy) increases parallelism and decreases communications.

Synthesizing: Constructing relational tables (in general, logical schemata), from a set of functional dependencies (see Functional dependency, Relation).

Tuple: A list of data values corresponding to a row of a relation. Each component of the tuple is a value extracted from the domain of a single column- attribute of the relation. In APPENDIX B, any row of relation R1 is a tuple of R1 (see Attribute, Domain, Relation).

Update: A transaction on a database that involves some writing (or deleting) operations. Data item values are added (or erased) to (from) the database.

View: Data needs of an organization or a user. In this work, a view is a set of functional dependencies.

APPENDIX B

THE RELATIONAL DATA MODEL

A RELATIONAL DATABASE EXAMPLE

HOSPITAL DATABASE FOR BLOOD TESTS

PATIENT_NO	NAME	AGE	WARD
4234	Stout	63	General Med.
5621	Rohkemper	34	Outpatient
1213	Sawdon	46	Outpatient
7784	Litkovitk	58	General Med.
2442	Kraphol	55	Intensive Care
3001	Buck	61	Intensive Care
6886	Rowlands	62	General Med.
2444	Kostedt	42	Ambulatory
3457	Rogulsky	53	General Med.
1972	Winter	39	Ambulatory

Relation R1

ORDER	NAME	TEST	RESULT	PHYSICIAN
01	Rohkemper	Glucose	70	Dr Newberry
03	Stout	Sodium	112	Dr Farah
05	Rohkemper	Potassium	2.5	Dr Akers
07	Kraphol	Glucose	50	Dr Morehouse
08	Kraphol	Sodium	160	Dr Morehouse
10	Rowlands	Glucose	80	Dr Jagiela
11	Rowlands	Chloride	25	Dr Jagiela
15	Winter	Glucose	100	Dr Yee
16	Stout	Glucose	60	Dr Andersland
17	Stout	Chloride	20	Dr Andersland

Relation R2

NAME	WARD
Stout	General Med.
Rohkemper	Outpatient
Sawdon	Outpatient
Litkovitk	General Med.
Kraphol	Intensive Care
Buck	Intensive Care
Rowlands	General Med.
Kostedt	Ambulatory
Rogulsky	General Med.
Winter	Ambulatory

R1' = R1[NAME,WARD]

PATIENT_NO	NAME	AGE	WARD
4234	Stout	63	General Med.
3001	Buck	61	Intensive Care
6886	Rowlands	62	General Med.

R1" = R1[AGE > 60]

ORDER	NAME	TEST	RESULT	PHYSICIAN
07	Kraphol	Glucose	50	Dr Morehouse
10	Rowlands	Glucose	80	Dr Jagiela
15	Winter	Glucose	100	Dr Yee
16	Stout	Glucose	60	Dr Andersland

R2' = R2[TEST = Glucose]

NAME	AGE	WARD	ORDER	TEST	RESULT	PHYSICIAN
Stout	63	General Med.	16	Glucose	60	Dr Andersland
Rowlands	62	General Med.	10	Glucose	80	Dr Jagiela

R12 = (R1"[NAME]R2')[NAME,AGE,WARD,ORDER,TEST,RESULT,PHYS.]

PATIENT_NO	NAME	AGE	WARD
4234	Stout	63	General Med.
6886	Rowlands	62	General Med.

R12' = R1"<NAME]R2'

APPENDIX C

THE MEDICAL RECORD DATABASE PROBLEM

A large health organization possesses four geographically separated sites: A main hospital with its own laboratory, a clinical laboratory that can provide additional products and services, a center for medical scientific research, and a department of health technology which comprises a statistical center and a section for general medical organization and development. The four sites of this organization are equipped with their individual computing facilities and have permanent access to a public fully connected local communication network. The staff members, at the four sites, share substantial informations about the medical records of the patients in the hospital. For that reason the organization plans to design an integrated distributed database, which will be referred to as the *medical record database* (MRDB).

User-provided FDs for MRDB		
ORDER	$\longrightarrow$	SERVICE
ORDER	$\longrightarrow$	RESULT
ORDER	$\longrightarrow$	TEST
ORDER	$\longrightarrow$	PATIENT_NO
ORDER	$\longrightarrow$	DATE
ORDER	$\longrightarrow$	NAME
ORDER,NAME	$\longrightarrow$	RESULT
TEST	$\longleftrightarrow$	TEST_TYPE,CODE_NO,TECHNIQUE
TEST,DATE,PATIENT_NO,NAME	$\longrightarrow$	RESULT
RESULT	$\longleftrightarrow$	TOTAL,PRECISION,ACCURACY,COST,TIME
PATIENT_NO	$\longrightarrow$	NAME
PATIENT_NO,NAME	$\longrightarrow$	AGE
PATIENT_NO	$\longrightarrow$	AGE
SERVICE	$\longrightarrow$	PHYSICIAN
SERVICE	$\longrightarrow$	WARD
PHYSICIAN	$\longrightarrow$	WARD
NAME	$\longrightarrow$	WARD

Table C.1. The user-provided functional dependencies (FDs).

Typical scheduled queries for MRDB		
Query i.d.	Given attributes	Target attributes
1	ORDER	PHYSICIAN,WARD
2	ORDER	RESULT
3	DATE,PATIENT_NO,TEST	ORDER
4	WARD	PATIENT_NO
5	ORDER	NAME,WARD
6	PHYSICIAN	ORDER,PATIENT_NO
7	NAME,DATE,TEST	RESULT
8	ORDER	SERVICE
9	PHYSICIAN	SERVICE,PATIENT_NO

Typical scheduled updates	
Number	New attributes
1	ORDER,TEST,RESULT,DATE
2	PATIENT_NO,NAME,AGE

Table C.2. Typical scheduled transactions for the MRDB.

Usage of MRDB transactions		
Site	Query	Update
1	1 X 2, 2 X 1, 3 X 1, 5 X 1, 8 X 1	none
2	1 X 1, 3 X 2, 9 X 1	none
3	5 X 3, 7 X 4, 8 X 4	none
4	4 X 1, 6 X 1, 9 X 1, 7 X 3	1 X 1, 2 X 2

Table C.3. Needs of each site in the MRDB computer network, and frequencies of occurrence of all the transactions.

f	$\|f\|$ *tuples*	$\bar{f}$ *bytes*
$f_{1,b}$	5075	9
$f_{1,c}$	5075	14
$f_{1,d}$	5075	8
$f_{1,e}$	5075	12
$f_{8,f}$	49	20
$f_{10,k}$	193	20
$f_{11,g}$	5075	44
$f_{12,i}$	84	19
$f_{12,j}$	84	6
$f_{15,k}$	810	20

Table C.4. Cardinalities and widths of the functional dependencies for the MRDB.

$\|f[A]\|$ *tuples*, where A is ...										
FD	ORDER	SER -VICE	TEST	PAT _NO	DATE	PHYS -ICIAN	RES -ULT	NAME	AGE	WARD
$f_{1,b}$	5075	48								
$f_{1,c}$	5075		76							
$f_{1,d}$	5075			84						
$f_{1,e}$	5075				632					
$f_{8,f}$		48				20				
$f_{10,k}$						20				10
$f_{11,g}$			76	84	632		6561			
$f_{12,i}$				84				84		
$f_{12,j}$								84	45	
$f_{15,k}$								84		10

Table C.5. Cardinalities of the functional dependencies, for the MRDB, when projected on attributes.

A	$\bar{A}$ *bytes*	
ORDER	4	
SERVICE	5	
TEST	TEST_TYPE	3
	CODE_NO	4
	TECHNIQUE	3
RESULT	NUMERIC	10
	PRECISION	10
	ACCURACY	10
	COST	5
	TIME	5
PATIENT_NO	4	
DATE	8	
NAME	15	
AGE	2	
PHYSICIAN	15	
WARD	5	

Table C.6. Attribute widths for the MRDB.

A	|DOM(A)|
ORDER	5075
SERVICE	48
TEST	76
RESULT	6561
PATIENT_NO	126
DATE	632
NAME	126
AGE	45
PHYSICIAN	28
WARD	10

Table C.7. Cardinalities of the domains for the MRDB.

APPENDIX D

INPUT FOR THE DISTRIBUTION ALGORITHMS

Number of sites: 4

Number of CRDUs: 13

Uncapacitated

User site	Query index	Query frequency
$t \in N$	k	$\nu_{t,k}$
1	2	1
1	8	1
3	8	4
4	4	1
1	1	2
1	5	1
2	1	1
3	5	3
3	7	4
4	6	1
4	7	3
1	3	1
2	3	2
2	9	1
4	9	1

Query	Partition	Cluster	# of FDs	FD 1	FD 2	FD 3
k	p	q	$\lvert q\rvert$	$h \in q$		
1	1	1	2	1	2	0
1	2	1	1	1	0	0
1	2	2	1	2	0	0
2	1	1	1	3	0	0
3	1	1	3	4	5	6
3	5	1	1	4	0	0
3	5	2	1	5	0	0
3	5	3	1	6	0	0
3	3	1	2	4	5	0
3	3	2	1	6	0	0
3	4	1	1	4	0	0
3	4	2	2	5	6	0
3	2	1	2	4	6	0
3	2	2	1	5	0	0
4	1	1	1	7	0	0
5	2	1	1	8	0	0
5	2	2	1	9	0	0
5	1	1	2	8	9	0
6	2	1	1	1	0	0
6	2	2	1	6	0	0
6	1	1	2	1	6	0
7	2	1	1	10	0	0
7	2	2	1	11	0	0
7	1	1	2	10	11	0
8	1	1	1	12	0	0
9	1	1	3	13	12	6
9	5	1	1	13	0	0
9	5	2	1	12	0	0
9	5	3	1	6	0	0
9	3	1	2	13	12	0
9	3	2	1	6	0	0
9	4	1	1	13	0	0
9	4	2	2	12	6	0
9	2	1	2	13	6	0
9	2	2	1	12	0	0

User site	Update index	Update frequency
s	k	$\nu_{s,k}$
1	1	0
1	2	0
2	1	0
2	2	0
3	1	0
4	1	1
3	2	0
4	2	2

Update index	FD	Update volume
k	h	B_k^h
1	1	500
1	3	500
1	4	500
1	5	500
1	6	500
1	8	700
1	10	700
1	12	700
2	6	450
2	7	450
2	8	450
2	9	450
2	10	900
2	11	900

Query	Partition	Cluster	Cluster	Data volume	Noncritical
k	p	q	r	A_k^{pqr}	Yes/No
1	1	1	1	450	1
1	2	1	1	400	1
1	2	1	2	5	0
1	2	2	1	4	1
1	2	2	2	10	1
2	1	1	1	1000	1
3	1	1	1	100	1
3	5	1	1	300	1
3	5	1	2	0	0
3	5	1	3	0	1
3	5	2	1	0	1
3	5	2	2	400	1
3	5	2	3	0	1
3	5	3	1	100	0
3	5	3	2	100	1
3	5	3	3	100	1
3	3	1	1	300	1
3	3	1	2	0	1
3	3	2	1	100	0
3	3	2	2	100	1
3	4	1	1	250	1
3	4	1	2	0	1
3	4	2	1	100	0
3	4	2	2	100	0
3	2	1	1	100	1
3	2	1	2	100	1
3	2	2	1	0	1
3	2	2	2	400	1
4	2	1	1	10000	1
5	2	1	1	0	1
5	2	1	2	100	0
5	2	2	1	0	1
5	2	2	2	100	1
5	1	1	1	100	1
6	2	1	1	0	1
6	2	1	2	10000	0

Query	Partition	Cluster	Cluster	Data volume	Noncritical
k	p	q	r	A_k^{pqr}	Yes/No
6	2	2	1	0	1
6	2	2	2	15000	1
6	1	1	1	15000	1
7	2	1	1	100	1
7	2	1	2	0	0
7	2	2	1	100	1
7	2	2	2	0	1
7	1	1	1	100	1
8	1	1	1	150	1
9	1	1	1	100	1
9	5	1	1	0	1
9	5	1	2	200	1
9	5	1	3	0	0
9	5	2	1	0	1
9	5	2	3	10000	1
9	5	3	1	0	1
9	5	3	2	100	1
9	5	3	3	100	1
9	3	1	1	100	1
9	3	1	2	10000	1
9	3	2	1	100	0
9	3	2	2	100	1
9	4	1	1	0	1
9	4	1	2	200	0
9	4	2	1	0	1
9	4	2	2	200	1
9	2	1	1	200	1
9	2	1	1	200	1
9	2	1	2	200	0
9	2	2	1	10000	1
9	2	2	2	0	1

For the capacitated model:

[c_{ij}] $(N \times N)$ matrix.

For all $(i,j) \in A$.

[K_{ij}] $(N \times N)$ matrix.

For all $(i,j) \in A$.

For both the capacitated and uncapacitated models:

[N^h] $(1 \times |H|)$ matrix.

(6 4 1 10 10 22 1 4 4 4 4 11 10)

APPENDIX E

OPTIMAL SOLUTION FOR THE MRDB
(Corresponding to the inputs of Appendix D)"

All the following variables are equal to one (the other variables are all equal to zero):

$X^{2,1}_{1,1,1}$, $X^{2,2}_{1,1,2}$, $X^{1,1}_{1,2,1}$, $X^{1,1}_{1,3,4}$, $X^{1,1}_{1,5,4}$, $X^{1,2}_{1,5,4}$, $X^{1,1}_{1,8,4}$, $X^{2,1}_{2,1,1}$,
$X^{2,2}_{2,1,2}$, $X^{1,1}_{2,3,4}$, $X^{1,1}_{2,9,4}$, $X^{2,1}_{3,5,4}$, $X^{1,1}_{3,7,4}$, $X^{1,2}_{3,7,4}$, $X^{1,1}_{3,8,4}$, $X^{1,1}_{4,4,4}$,
$X^{2,1}_{4,6,4}$, $X^{1,1}_{4,9,4}$, $X^{2,1}_{4,7,4}$, Y^{1}_{1}, Y^{3}_{1}, Y^{2}_{2}, Y^{1}_{4}, Y^{4}_{4}, Y^{5}_{4},
Y^{6}_{4}, Y^{7}_{4}, Y^{8}_{4}, Y^{9}_{4}, Y^{10}_{4}, Y^{11}_{4}, Y^{12}_{4}, Y^{13}_{4}, $X^{2}_{1,1}$,
$X^{1}_{1,2}$, $X^{1}_{1,3}$, $X^{1}_{1,5}$, $X^{1}_{1,8}$, $X^{2}_{2,1}$, $X^{1}_{2,3}$, $X^{1}_{2,9}$, $X^{2}_{3,5}$, $X^{1}_{3,7}$,
$X^{1}_{3,8}$, $X^{1}_{4,4}$, $X^{2}_{4,6}$, $X^{1}_{4,9}$, $X^{2}_{4,7}$.

BIBLIOGRAPHY

BIBLIOGRAPHY

[AHO77] Aho, A. V., Beeri, C., and Ullman, J. D., "The theory of joins in relational databases", *Proc.29th, IEEE Symp.,* Foundations of Comp.Sc. 1977.

[APER81] Apers, P. M. G., "Redundant allocation of relations in a communication network", *Proc. of the 5th Berkeley Workshop on Distributed Data Management and Computer Networks,* Lawrence Berkeley Lab. 1981.

[APER83] Apers, P. M. G., Hevner, A. R., and YAO, S. B., "Optimization algorithms for distributed queries", *IEE Trans. on Software Engin., Vol. SE-9,* No. 1, Jan. 1983.

[FEDA81] Al-Fedaghi, S., and Scheuermann, P., "Mapping considerations in the design of schemas for the relational model", *IEEE Trans. on Soft. Eng., Vol. SE-7,* No 1, Jan 1981.

[ARMS74] Armstrong, W. W., "Dependency structures of data bases relationships", *Information Processing 74,* North-Holland Publ. co., Amsterdam, 1974 pp 580-583.

[BEER80] Beeri, C., "On the membership problem for functional and multivalued dependencies in relational databases", *ACM TODS, Vol.5,* No 3, Sept. 1980.

[BEER81] Beeri, C., Fagin, R., Maier, R., Mendelzon, A., Ullman, J., and Yannakakis, M., "Properties of acyclic database schemes", *Proceedings of the ACM Symposium on Principles of Database Systems,* 29-31 March 1982.

[BERN76] Bernstein, P. A., "Synthesizing third normal form relations from functional dependencies", *ACM TODS, Vol.1,* No 4, Dec. 1976.

[BISK79] Biskup, J., Dayal, U., Bernstein, P. A., "Synthesizing independent database schemas", ACM SIGMOD, *International Conference on the Management of Data,* 1979.

[BOOT81] Booth, G. M., *The distributed system environment some practical approaches,* Mc Graw-Hill 1981.

[BUCK79] Buckles, B. P., Hardin, D. M., "Partitioning and allocation of logical resources in a distributed computing environment", *Distributed System Design, Tutorial,* IEEE Catalog No EHO261-1 1979.

[CASE72] Casey, R. G., "Allocation of copies of a file in an information network", *AFIPS Conference Proceedings, vol.41,* part I, 1972.

[CHAN77] Chandy, K. M., "Models of distributed systems", *Proceedings VLDB Conference,* Oct. 1977.

[CHEN80] Chen, P. P.-S., and Akoka, J. "Optimal Design of Distributed Information Systems", *IEEE Trans. on Computers, Vol. C-29,* No 12, Dec. 1980.

[CHU73] Chu, W. W., "Optimal file allocation in a computer network", *Computer-Communications Network,* N.Abramson and F.Kuo (eds), Prentice-Hall, Englewood Cliffs, N.J., 1973.

[CHU79] Chu, W. W., and Hurley, P., "A model for optimal query processing for distributed databases", *Digest of Papers, COMCON,* Spring 1979.

[CHU76] Chu, W. W., "Performance of file directory systems for databases in star and distributed networks", *AFIPS Conference Proceedings, Vol. 45,* 1976 NCC.

[CHUN83] Chung, C.-W., *A query optimization in distributed database systems,* PhD Thesis, The University of Michigan, 1983.

[CODD79] Codd, E. F., "Extending the database relational model to capture more meaning", ACM SIGMOD *International Conf. on Management of Data,* p 161, 1979.

[CODD72] Codd, E. F., "Further normalization of the database relational model", *Database Systems,* R.Rustin, eds, Prentice-Hall 1972.

[CCA80] Computer Corporation of America, *A distributed database management system for command and control applications,* Technical Report CCA-80-04, Jan.30 1980.

[DATE78] Date, C. J., *An introduction to database systems,* Addison-Wesley 1978.

[DELO78] Delobel, C., "Normalization and hierarchical dependencies in the relational data model", *ACM TODS, Vol.3,* No 3, Sep 1978.

[DORA77] Doray, M., "Le teletraitement", Societe Nationale des Chemins de Fer- Service Informatique (Paris), Division IAT, May 1977.

[ESWA74] Eswaran, K. P., "Placement of records in a file and file allocation in a computer network", *Information Processing 74,* IFIPS, North Holland Publishing Co., 1974.

[ETCH77] Etcheberry, J. "The Set-Covering Problem: A New Implicit Enumeration Algorithm", *Operations Research, Vol. 25,* No 5, Sept.-Oct. 1977.

[FAGI77] Fagin, R., "Multivalued dependencies and a new normal form for relational databases", *ACM TODS, vol.2,* no 3, Dec. 1977.

[FAGI82] Fagin, R., Mendelzon, A. O., and Ullman, J. D., "A simplified universal relation assumption and its properties", *ACM TODS, Vol. 7,* No. 3, Sep. 1982.

[FISH80] Fisher, M. L., and Hochbaum, D. S., "Database location in computer networks", *Journal of ACM, vol.27,* no 4, Oct. 1980.

[FRY78] Fry, J. P., and Teorey, T. J., "Design and performance tools for improving database usability and responsiveness", *Databases: Improving usability and responsiveness,* Academic Press 1978.

[GARE79] Garey, M. R., and Johnson, D. S., *Computers and intractability- A guide to he theory of NP-completeness,* Freeman 1979.

[HAMM79] Hammer, M., and Niamir, B., "A heuristic approach to attribute partitioning", *Proc. ACM SIGMOD, Int. Conf. on the Management of Data,* May 1979, pp 93-101.

[HEVN79] Hevner, A. R., and Yao, S. B., "Query processing in distributed database systems", *IEEE Trans. on Soft. Engin., Vol.SE-5,* No 3, May 1979.

[HIMM72] Himmelblau, D. M., *Applied Nonlinear Programming,* Mc Graw Hill 1972.

[HOLL75] Holland, J. H., *Adaptation in Natural and Artificial Systems,* The University of Michigan Press 1975.

[HOUS79] Housel, B. C., Waddle, V., Yao, S. B., "the functional dependency model for logical database design", IEEE 1979.

[KAMB78] Kambayashi, Y., "Equivalent key problem of the relational database model", *Mathematical studies of information processing, Proceedings,* Kyoto, Japan 1978. Ed. by E.K.Blum, M.Paul and S.Takasu, Springer-Verlag.

[KHAB80] Khabbaz, N. G., *A combined communication network design and file allocation for distributed databases,* PhD thesis, The University of Michigan, Feb. 1980.

[LAWL76] Lawler, E., *Combinatorial optimization,* Holt, Rinehart and Winston 1976.

[LEVI75] Levin, K. D., and Morgan, H. L., "Optimizing distributed databases-A framework for research", *AFIPS Conference Proceedings, vol.44,* 1975 NCC.

[LOZI78] Lozinskii, E. L., "Performance consideration in relational database design", *Databases: Improving usability and responsiveness,* Academic Press Inc., 1978.

[LOZI80] Lozinskii, E. L., "Construction of relations in relational databases", *ACM TODS, Vol.5,* No 2, June 1980.

[MAHM76] Mahmoud, S., and Riordon, J. S., "Optimal allocation of resources in distributed information networks", *ACM TODS, vol.1,* no 1, Mar.1976.

[MAIE80] Maier, D., "minimum covers in the relational database model", *Journal of ACM, Vol. 27,* No 4, Oct. 1980.

[MAIE83] Maier, D., and Ullman, J. D., "Maximal objects an the semantics of universal relation databases", *ACM TODS, Vol. 8,* No. 1, Mar. 1983.

[MORG77] Morgan, H. L., and Levin, K. D., "Optimal program and data locations in computer networks", *Communication fo the ACM, Vol.32,* No 5, May 1977.

[MURT76] Murty, K., *Linear and combinatorial programming,* John Wiley and sons Inc., 1976.

[MURT81] Murty, K., *Network flow algorithms,* lecture notes, Department of Industrial Engineering and Operations Research, The University of Michigan, 1981.

[MUR77] Murtagh, B. A., and Saunders, M. A. *MINOS A Large-Scale Nonlinear Programming System- User's Guide,* Technical Report Sol. 77-9 Feb. 1977, Systems Optimization Lab., Department of Operations Research, Stanford University.

[MUR78] Murtagh, B. A., and Sauders, M. A. "Large scale linearly constrained optimization", *Mathematical Programming, Vol. 14,* p. 41-72, 1978.

[NAVA76] Navathe, S. B., and Fry, J. P., "Restructuring for large databases: Three levels of abstraction", *ACM TODS, Vol.1,* No 2, Jun 1976.

[PALM79] Palmer, D. F., "Distributed computing system design at the subsystem-network level", *Conf. VLDB,* IEEE, 1979.

[RAM79-a] Ramamoorthy, C. V., and Wah, B. W., "Data management in distributed data bases", *Proc. National Conf.,* AFIPS Press, 1979.

[RAM79-b] Ramamoorthy, C. V., and Wah, B. W., "The placement of relations on a distributed relational database", *The 1st International Conference on Distributed Computing Systems,* Huntville, Alabama, Oct. 1-5, 1979.

[RISS77] Rissanen, J., "Independent components of relations", *ACM TODS, Vol.2,* No 4, Dec 1977.

[ROSE81] Rosenthal, A. S., "Note on the expected size of a join", *SIGMOD Record, Vol. 11,* No. 4, Jul. 1981.

[SALK75] Salkin, H. M., *Integer programming,* Addison-Wesley 1975.

[TANE81] Tanenbaum, A. S., *Computer networks,* Prentice-Hall 1981.

[ULLM80] Ullman, J. D., *Principles of database systems,* Computer Science Press, 1980.

[ULLM82] Ullman, J. D., "The U.R. Strikes Back", *Proceedings of the ACM Symp. on Principles of Database Syst.,* Los Angeles, California March 1982.

[WANG75] Wang, C. P., and Wedekind, H. H., "Segment synthesis in logical database design", *IBM J. Res. Dev., Vol. 19,* No 1, Jan. 1975.

[WONG77] Wong, E., "Retrieving dispersed data from SDD-1: A system for distributed databases", *Proc. of the 2nd Berkeley Workshop on distributed data management and computer networks,* Lawrence Berkeley Lab., May 1977.

[WONG81] Wong, E., "Dynamic re-materialization: Processing distributed queries using redundant data", *Proc. of the 5th Berkeley Workshop on Distributed data management and computer networks,* Lawrence Berkeley Lab. 1981.