

EFFICIENT ABSTRACT IMPLEMENTATIONS FOR
RELATIONAL DATA STRUCTURES

by

Nicola Santoro

RESEARCH REPORT CS-80-21

University of Waterloo
Department of Computer Science
Waterloo, Ontario, Canada

April 1980

EFFICIENT ABSTRACT IMPLEMENTATIONS FOR
RELATIONAL DATA STRUCTURES

by

Nicola Santoro

A thesis

presented to the University of Waterloo

in partial fulfillment of the

requirement for the degree of

Doctor of Philosophy

in


Computer Science

Waterloo, Ontario, Canada, 1979


© Nicola Santoro

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.


Signature

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.


Signature

The University of Waterloo requires the signature of all persons using or photocopying this thesis. Please sign below, and give address and date.

Efficient Abstract Implementations for Relational Data Structures

Nicola Santoro

ABSTRACT

The goal of this thesis is to present some foundations for choosing an efficient abstract structure for a relational data structure described by means of its conceptual model. Assuming that data inter-relationships can be expressed as binary relations, the time and space complexity of abstract structures are defined and conditions for an abstract structure to be efficient and optimal, within a constant factor, are stated.

As a preliminary step to developing this theory, a formalism for describing data structures is developed, first in terms of the algebra of binary relations and then as an algebra over relational graphs. The theory is then applied to hierarchical structures, that is, those corresponding to relational trees. For these we present algorithms to revise a given abstract structure into one which improves the time and/or space efficiency. Where possible, these results are also extended to nonhierarchical relational graphs.

The proposed techniques are illustrated by means of examples chosen from data base design. Applications to communication network design and to picture processing are also described.

Key Words and Phrases:

data structures, formal data models, computational complexity,
binary relations, efficient and optimal representations,
hierarchical structures, data base design.

Acknowledgements

First of all, I would like to thank my supervisor Frank Tompa for his encouragement, suggestions and advice. His constant positive attitude and his friendship have proven invaluable.

I would like to thank Fabrizio Luccio and Ian Munro for generating my interest in computational complexity and information structures, for the many helpful discussions and their friendship.

Many people have contributed in several ways to the completion of this thesis: to all of them my gratitude. In particular, I would like to thank the members of my committee: Wes Graham, Tiko Komeda, Ian Munro and Ken Sevcik; my friends and colleagues: Raúl Ramirez, Hendra Suwanda and Denis Therien; my wise friend Bruno Forte; all the people who made my stay in Waterloo an enriching experience: the alternative collectives, Dumont, Integrated Studies, K.F. Gauss Foundation, CKMS, the Emergency of St. Joseph Hospital, Darlene, Ino, Jane, Saverio, Herman, and many, many others; Mirella - in spite of whom this thesis was started, Antimo, Giovanni, Luciana, i Sorrenti Tutti and the many friends in Chieti and Pisa; and, last but not least, Wiz - in spite of whom this thesis has been completed.

I wish to acknowledge the financial support of the University of Pisa, the University of Waterloo, the Natural Sciences and Engineering Research Council and the Comitato per la Matematica del Consiglio Nazionale delle Ricerche.

Finally, I wish to thank my parents and my children, Monica and Noël, to whom this thesis is dedicated.

TABLE OF CONTENTS

	<u>Page</u>
1. INTRODUCTION	
1.1 The Problem.....	1
1.2 Related Work.....	7
1.3 Outline Of The Thesis.....	9
2. RELATIONAL ABSTRACT STRUCTURES	
2.1 Fundamental Concepts.....	12
2.1.1 The Model.....	12
2.1.2 Relational Graphs.....	17
2.1.3 Efficiency Parameters.....	22
2.2 Fundamental Properties.....	26
2.2.1 Equivalence Properties.....	28
2.2.2 Constructing A Perfect Implementation.....	32
2.2.3 Hierarchical Conceptual Models.....	37
3. STAR GRAPH APPROACH	
3.1 Introduction.....	44
3.2 Constructing The Star Graph.....	51
3.3 Complexity Of The Star Graph.....	53
3.4 Constructing The Complement.....	58

	<u>Page</u>
4. REDUNDANT GRAPH APPROACH	
4.1 Introduction.....	66
4.2 Constructing An Efficient Redundant Implementation.....	68
4.2.1 General Case.....	68
4.2.2 Special Cases.....	76
5. GENERAL CONCEPTUAL MODELS	
5.1 Introduction.....	80
5.2 Heuristic Methods.....	82
5.3 Relational Graphs With One Circuit.....	90
6. CONCLUSIONS AND APPLICATIONS	
6.1 Summary.....	97
6.2 A Final Example.....	102
6.3 Other Applications.....	112
6.3.1 Picture Processing.....	112
6.3.2 Communication Networks.....	114
6.4 Conclusions And Extensions.....	116
References.....	118
APPENDICES	
A. Extended Algebra of Binary Relations.....	125
B. Properties of Relational Graphs.....	128
C. Data For The Final Example.....	131

CHAPTER 1

INTRODUCTION

1.1 THE PROBLEM

1.2 RELATED WORK

1.3 OUTLINE OF THE THESIS

Un tuffo dove l'acqua

è più blu.

Niente di più.

FORMULA TRE

Do you understand, Mr. Lamont?

I do indeed - you are quite
right.

FLANN O'BRIAN

Ταυτα δε γέ φαίνεται

άγωγα προς αλήθειαν

PLATO

CHAPTER 1

INTRODUCTION

1.1 The Problem

In recent times, the desirability of attacking problems at several levels of abstraction has appeared in different fields. For example the structured programming approach [Ben, Dij, Hoa, McGo, Wir], the use of abstract data types [Gut, Lis], and the introduction of an open system architecture [ISO], all used to improve the modularity of programs and therefore simplify their design, are the major examples of this trend in the areas of programming languages, data management and data communication. In data structure design, many authors have each emphasized only one or two levels of abstraction [Dim, Ka, La, McC, Me, Mi, Sel, Su]. A detailed description of the problem and a survey of suggestions previously proposed are contained in [To2], where five levels of data refinement are distinguished:

1. data reality: the real world, data object.
2. conceptual model: a model of the real world where (only) the properties relevant to the specific application(s) are contained.
3. abstract structure: a refinement of the conceptual model in which only some properties are made explicit, the others derivable indirectly through computation.
4. storage structure: a model of the abstract structure in terms of

representations for each of the data type occurrences.

5. primitive encoding: the computer representation that concerns the physical devices involved.

To each level of abstraction for data, we can associate a level of abstraction for the operations performed on the data. Thus we can distinguish the following levels:

1. the goals of the problem in terms of real objects.
2. the specification of a solution in terms of conceptual models.
3. the algorithm incorporating the specifications.
4. the program implementing the algorithm.
5. the object code implementing the program to operate in a particular machine environment.

The relationships among different levels of abstractions are hierarchical, that is, any modification at one level implies a change in lower levels but should not necessitate a change in any upper level. At the same time, there is a dependency between levels of abstraction of data and levels of operations on data. For example, at the storage structure level if we use lists instead of hash tables to represent a symbol table then we have to implement different programs for all the algorithms that manipulate the table.

The problem of efficiency has been examined at different levels of abstraction and at the transition from one level to the next [Low, To1, To2]. Although it is widely recognized that improvements to an algorithm

are more significant than improvements to a corresponding program, less research has been devoted to the transition from a conceptual model to an abstract structure than to the one from an abstract to a storage structure.

The aim of this thesis is to narrow this research gap by analyzing the first transition; that is, to analyze the refinement process that leads to express directly, in the abstract structure, only some of the properties contained in the conceptual model, the others being available through computation. We will discuss the concept of efficiency (induced by such computation), as well as the notions of implementation, semantic equivalence and integrity. In just one sentence, we will examine the problem of choosing an efficient abstract structure implementing a given conceptual model.

In terms of data base design, this problem is to decide which data relationships should be stored explicitly in order to maximize the efficiency of the system. Although the formalism introduced in this thesis can be used to describe complex structures arising in several different (and often unrelated) large-scale software applications, we will preferably refer in our examples to problems and situations arising from data base design. The main reason lies in the immediate correspondence between the five-levels view of the data used in this thesis and the views used in all the important data base models (for example, the ANSI/SPARC architecture [Ts1], the infological/detalogical model

[Lan, Sun], and the role model [Bac]).

We will assume that all the relationships among data can be expressed as binary relations. This apparent restriction has been defended elsewhere, since several models for data structure design use only binary relations [Ab, Ne, Br, Fel, Ash, Le, Se2]. Furthermore, as discussed later, any n-ary relation can be expressed as a set of binary relations; therefore this assumption does not limit the generality of the formalism, although efficient solutions for binary relations may not indicate efficient solutions for n-ary relations.

Under this assumption, a conceptual model can be defined by a set of data elements and a set of relevant relations on the data. Thus an abstract structure implementing the conceptual model can be characterized by a set of relations whose transitive closure contains the whole set of relations in the conceptual model. Using this formalism we can precisely describe conceptual models and abstract structures and analyze the efficiency of structures implementing a conceptual model.

The parameters introduced in order to evaluate the efficiency of an abstract structure take into account the space and the time complexity of the structure. We measure the space efficiency of an abstract structure by the numbers of explicitly stored relations, and the time efficiency by the number of relations that must be composed in order to answer a query (retrieval time). This is done in order to use complexity measures that are general and defined at the considered level

of abstraction, and at the same time are "realistic" (i.e. immediately translatable to complexity parameters used at lower levels of abstraction). For example, if we are in a paged environment where exactly one relation fits on a page (or a fixed number of pages), the number of relations describes how many pages are needed to store the information, and the retrieval time measures the number of paging operations required to answer a query. Using these parameters we will state conditions for an abstract structure to be efficient and optimal (within a constant factor).

In this thesis we have obtained several original results, which, using a dichotomic view of reality, can be partitioned into "theoretical" and "practical" ones. The first basic theoretical result is the formal definition of a useful framework to analyze the transition from the conceptual model level to the abstract structure level. As part of that framework, the most interesting theoretical result is the determination of a basic equivalence between two general problems, as follows. We prove that there exists a basic class of implementations for any conceptual model, and that to solve the efficiency problem (i.e. the problem of finding an efficient abstract structure implementing a given conceptual model) is equivalent to transforming a member of that class into another "semantically equivalent" abstract structure. This result gives us a constructive way to attack the efficiency problem and to obtain practical results.

In the realm of practical results, we show that, for any given conceptual model, there are two important abstract structures implementing it. The first structure, I^* , requires $O(n^2)$ space and allows $O(1)$ retrieval time, where n is the number of domains in the model; the other structure, I_0 , needs $O(n)$ space and $O(n)$ time. Both structures are limiting ones; in fact we prove that it is impossible to implement a conceptual model using less space than I_0 or achieving better retrieval time than I^* . Therefore we have directed our attention to structures that achieve a better trade-off between space and time. We show how to construct structures that allow $O(1)$ retrieval time with only $O(n \log n)$ explicitly stored relations. The advantage of these structures with respect to I_0 depends on the actual cost of space and time in the particular application. For instance, in a paged environment, the trade-off is between whether it is more convenient to have $O(n)$ pages but $O(n)$ page-in/page-out operations on average when we process a query, or rather to have $O(n \log n)$ pages with a constant number of paging operations.

The second class of abstract implementations considered in the thesis is perhaps the more important one. We show that it is possible, under certain conditions, to construct abstract implementations requiring $O(n)$ relations (i.e. the minimum amount of space needed to implement a conceptual model) and achieving $O(1)$ retrieval time (i.e. the minimum possible retrieval time). We analyze the conditions under which such "optimal" implementations can be constructed and show that these conditions depend on the nature of the relations in the model.

All the above results are proved for an important class of conceptual models, i.e., the hierarchical models. Extensions of these results to more general models are discussed. Many problems arising from data base design and other applications are inherently hierarchical in nature (e.g. hierarchical data bases [Tsi2], networks of binary relations associated with n-ary constraints [Mol], some CODASYL-compatible data bases [Deh], etc.).

As a final contribution, the thesis also contains some original results in graph theory, primarily a partitioning existence theorem on trees. We have introduced relational graphs (i.e. directed edge-labelled multi-graphs) as a useful mechanism to describe abstract structures. After transforming the algebra of binary relations into an algebra over relational graphs, graph-theoretic results have been used to prove properties of abstract structures.

1.2 Related Work

As previously noted, the transition from conceptual models to abstract structures has been largely ignored, mostly because of the absence of a formalism in which the problem could be defined and analyzed. Although many researchers have more or less explicitly stated the need for such investigation [Bub, Ham, Ka, To2], no systematic research, to our knowledge, has been devoted to this problem. This absence of theoretical work is reflected in the practical applications. For example (with the possible exception of the study of normalization [Bee]),

in data base design there has been a fatalistic approach toward the abstract structure level: designers are given one abstract structure and their only problem is how to implement it; after formulating a correct abstract structure, no further consideration is given as to whether the structure is the most efficient statement for the problem.

While there is practically no research done on the transition from conceptual models to abstract structures, several studies have been pursued at the next lower level of abstraction. Typically, the algebraically-based formalisms [Gut] for data-types, and the techniques based on graph theory [Cha, Got, Har, Maj, Mun, Ray] assume data abstractions as a "datum", without any reference to conceptual models or to the refinement process that led to the determination of such structures.

Many sub-problems analyzed in this thesis have been studied in different contexts and/or from different perspectives. For example, several studies have been done in the field of the semantic equivalence of data structures [Bil, McGe], and in the analysis of semantic-preserving transformations on simple structures [Da], Ros]. Techniques similar to the redundant graph approach have been studied to increase reliability, and it is worth noting that the notion of a star graph introduced in this thesis is implicit in some of Senko's writing [Sen2].

Finally, research on problems similar to the one studied in this thesis has been done in fields other than data structure design; noticeable

examples can be found in picture processing [Mol, Sa1], and in communication networks design [Cer, Sa2].

The related work will be described in more detail where appropriate in the remainder of the thesis.

1.3 Outline Of The Thesis

Following this introduction, we present the formalism to define conceptual models and abstract structures. In Chapter 2, we also introduce relational graphs as a useful mechanism to describe abstract structures and analyze their complexity; define time and space efficiency of abstract structures and state conditions for an abstract structure to be efficient and optimal (within a constant factor); state a fundamental equivalence which will enable us to attack the efficiency problem; and characterize a particular class of conceptual models, the hierarchical conceptual models.

In Chapters 3 and 4 we present two methods to derive efficient abstract implementations for hierarchical conceptual models, analyze the properties of such implementations, and derive bounds on their complexity. Then in Chapter 5 we extend these results to general conceptual models, and present techniques to solve the efficiency problem for such models.

Finally, in Chapter 6 we summarize the results obtained in this thesis, describe through a detailed example some of the proposed

techniques, and present some problems that can be reformulated and solved using the techniques discussed in this thesis, even though these problems arise from fields unrelated to data structure design. Chapter 6 also contains a brief survey of open problems and directions for further research.

Throughout the thesis, we will assume that the reader has some familiarity with the algebra of binary relations and with the properties of digraphs (see, for example [Ber, Gra]). Brief descriptions of both the classical and the non-standard terminology and definitions used in this thesis are contained in Appendices A and B. Appendix C contains the data used in the extended example described in Chapter 6.

CHAPTER 2

RELATIONAL ABSTRACT STRUCTURES

2.1 FUNDAMENTAL CONCEPTS

2.1.1 The Model

2.1.2 Relational Graphs

2.1.3 Efficiency Parameters

2.2 FUNDAMENTAL PROPERTIES

2.2.1 Equivalence Property

2.2.2 Constructing A Perfect Implementation

2.2.3 Hierarchical Conceptual Models

CHAPTER 2

RELATIONAL ABSTRACT STRUCTURES

2.1 Fundamental Concepts

2.1.1 The Model

Let us describe the model we are using in our research. A conceptual model has been defined above as a model of the real world in which the properties relevant to the problem are contained. Given a set of data D , let R_D denote the set of all possible binary relations between subsets of D (see also Appendix A).

Definition 1. A conceptual model I is a couple $I = (D, R)$ where D is a non-empty set of data elements and $R \subseteq R_D$ is the set of all the data interrelationships relevant to the problem.

Since all relationships of interest to the application being modeled are, by definition, included explicitly in R , then a query on I is equivalent to a request for the value of some relation r in R ; thus, in our model, a query on I is any relation $r \in R$. If $ld[r]$ and $rd[r]$ represent the left and right domain of r , respectively, then let $C(R) = \{d_j \in D \mid r \in R (d_j = rd[r] \text{ or } d_j = ld[r])\}$ denote the set of domains of I . We will assume that each data element appears at

at least in one domain.

A (relational) abstract structure is a refinement of the model in which only some relations are made explicit, the others derivable indirectly through computation.

Definition 2. An abstract structure I_i implementing the conceptual model $I = (D, R)$ is a triple $I_i = (D_i, R_i, \psi_i)$ where

1. $D_i \supseteq D$
2. $(R_i)^* \supseteq R$ where $(R_i)^*$ denotes the set of all distinct relations that can be obtained through composition of elements of R_i .
3. $\psi_i : R \rightarrow S(R_i)$ is a query mapping defined as follows:
for all $r \in R$, $\psi_i(r)$ is a sequence $q \in S(r_i)$ of minimum length such that $q \equiv r$ (i.e., q and r are equivalent, see Appendix A).
4. There is a one-to-one correspondence between $C(R_i)$ and $C(R)$, $(C(R) \Leftrightarrow C(R_i))$.

In this definition we use inclusion instead of equality to show that supplementary data elements and relations may be introduced in an implementation. In general, the representation of a binary relation implicitly contains the representation of its inverse, e.g. in a Boolean matrix representation, the inverse of a relation is represented by the transpose

of the given matrix; in the following we will assume that if $r \in R$ then $r^{-1} \in R$.

Let us remark that condition 2 in the above definition does not imply the existence of the query mapping ψ_i . For example, let $R = \{a,b,c,d,e\}$ with $c = a \cdot b$ and $b = a^{-1} \cdot d \cdot e$; the set $R_i = \{a,d,e\}$ is such that $(R_i)^* \supseteq R$, but it is easy to see that no query mapping is constructable using such set.

Let us observe that, for a given conceptual model $I = (D,R)$, there is always at least one abstract structure implementing I . In fact, the abstract structure (D,R,\mathcal{I}) , where \mathcal{I} is the identity function, is obviously an implementation of I . Later in this paper we will make use of this observation and refer to (D,R,\mathcal{I}) as I^* .

When equality holds for the sets of data elements and $(R_i)^* = (R)^*$, we will say that the relational abstract structure is a perfect implementation of the conceptual model. Let $G(I)$ denote the set of all abstract structures implementing I , and $P(I)$ the set of the perfect implementations of I .

Among all the relational abstract structures which perfectly implement a conceptual model, we can characterize a set $\text{Basis}(I)$ to be the minimal perfect implementations, as follows:

$$I_i \in \text{Basis}(I) \text{ iff } I_i \in P(I) \text{ and } \forall r \in R_i \ (R_i - r)^* \supseteq R$$

Informally, a relational abstract structure I_i in the basis is such that no substructure of I_i implements I . The basis is possibly an empty set, and is unique for a given conceptual model; in fact, in many cases it consists of only one element. Our interest in $\text{Basis}(I)$ is not purely based on academic grounds: since we are concerned with efficiency, the number of relations needed to implement a conceptual model is (as we will discuss later) an important space-complexity parameter. When non-empty, $\text{Basis}(I)$ consists of implementations of I that do not contain redundant relations to be stored explicitly; therefore it represents space-efficient ways of implementing I . A set inclusion diagram of $G(I)$, $P(I)$ and $\text{Basis}(I)$ is shown in Figure 2.1.

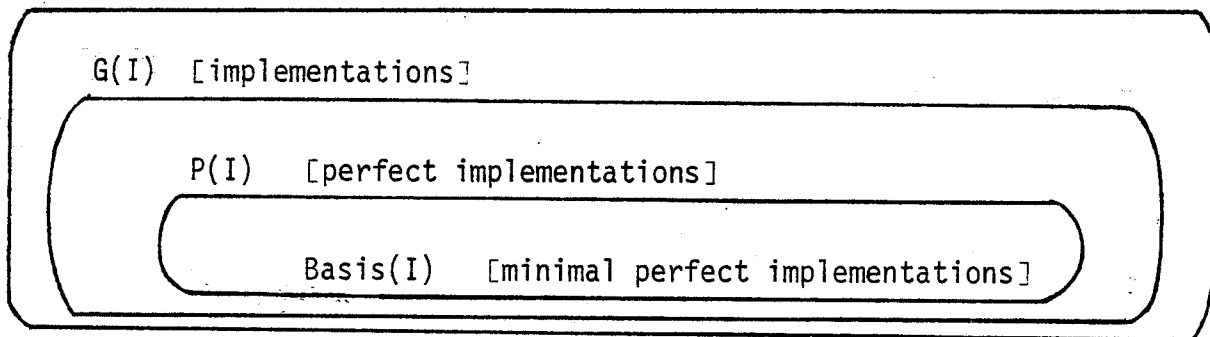


Fig 2.1 Set inclusion diagram of the abstract structures implementing I .

The analogy between the above terminology and the one used in relational data base systems [Cod, Date] is not accidental. Let us further note the similarity between the query mapping ψ_i and the search path in independent accessing models [Gho, Sch]: the user's view of the data is defined independently of the structure used for the implementation. To the users, all relations appear to be equally accessible (as in the

conceptual model), although in the implementation there exists a retrieval mechanism (the query mapping), hidden to the users, to access the relations.

Before proceeding, we will attempt to illustrate the central concepts by means of a trivial example.

Example 1.

Let $I = (D,R)$ be as follows:

$$D = \{u,m,a,b,c,s_1,\dots,s_5,t_1,\dots,t_5\}$$

$$R = \{t,e,p,ep,te,tep,t^{-1},e^{-1},p^{-1},ep^{-1},te^{-1},tep^{-1}\}$$

where $ep = e \cdot p$, $te = t \cdot e$, and $tep = t \cdot e \cdot p$, and the relations are as follows:

$$t = \{(u,a), (u,b), (m,c), (\delta,\delta)\}$$

$$e = \{(a,s_1), (a,s_2), (b,s_3), (c,s_4), (c,s_5), (\delta,\delta)\}$$

$$p = \{(s_1,t_1), (s_2,t_2), (s_3,t_3), (s_4,t_4), (s_5,t_5), (\delta,\delta)\}$$

$$ep = \{(a,t_1), (a,t_2), (b,t_3), (c,t_4), (c,t_5), (\delta,\delta)\}$$

$$te = \{(u,s_1), (u,s_2), (u,s_3), (m,s_4), (m,s_5), (\delta,\delta)\}$$

$$tep = \{(u,t_1), (u,t_2), (u,t_3), (m,t_4), (m,t_5), (\delta,\delta)\}$$

The set of domains $C(R)$ is given by $C(R) = \{d_1,d_2,d_3,d_4\}$

where $d_1 = \{u,m\}$, $d_2 = \{a,b,c\}$, $d_3 = \{s_1,s_2,s_3,s_4,s_5\}$ and

$d_4 = \{t_1,t_2,t_3,t_4,t_5\}$.

A perfect implementation of I is $I_1 = (D,R_1,\psi_1)$ where

$R_1 = \{tep,e,p\}$ and ψ_1 is defined by:

$$\begin{aligned} e &\longrightarrow \langle e \rangle ; & t &\longrightarrow \langle tep p^{-1} e^{-1} \rangle ; \\ p &\longrightarrow \langle p \rangle ; & te &\longrightarrow \langle tep p^{-1} \rangle ; \\ ep &\longrightarrow \langle e p \rangle ; & tep &\longrightarrow \langle tep \rangle . \end{aligned}$$

It is easy to show that $I_1 \in \text{Basis}(I)$. Other elements in $\text{Basis}(I)$ are I_2, I_3, I_4, I_5, I_6 where $R_2 = \{te, e, p\}$, $R_3 = \{t, e, p\}$, $R_4 = \{t, ep, p\}$, $R_5 = \{tep, ep, p\}$, $R_6 = \{te, ep, e\}$.

2.1.2 Relational Graphs

There have been several recent papers describing abstract structures in terms of graphs and graph grammars [Cha, Maj, Mo2, Ray, Ros, Tsu]. In fact, the notation of graphs is very useful for visualizing properties of binary relations, and therefore we will now transform the notions of relational abstract structures into their images in relational graphs.

A directed edge-labelled multigraph is a triple (V, L, E) where V and L are arbitrary non-empty sets of elements, named vertices and labels, respectively, and $E \subseteq V \times V \times L$ is a named set of edges. Given $x, y \in V$, $z \in L$, if there is an edge $e \in E$ from vertex x to vertex y having label z , we denote this $e = (x, y; z)$ and call x the start vertex, y the end vertex and z the label of e . (Further relevant graph theoretic concepts are included in Appendix B.)

A relational graph is a directed edge-labelled multi-graph in which the vertices represent domains and the labels name binary relations.

We claim there is a (natural) isomorphism between relational abstract structures and relational graphs, where the image of (D_i, R_i, ψ_i) is the graph (V_i, L_i, E_i) in which the vertices represent the domains in $C(R_i)$, the labels represent the relations in R_i , and a directed edge labelled r_i has start vertex d_j and end vertex d_k if, and only if, the relation r_i is between the domains d_j and d_k ($[r_i] = (d_j, d_k)$). The relational graph corresponding to the abstract structure I_1 in Example 2.1 is shown in Figure 2.2.

Because the notion of query mapping is central to the calculation of time efficiency (as it will be discussed later), we now relate a sequence of relations and inverse relations in R_i to its image in a relational graph. Thus, if $(d_1, d_2; r)$ is an edge in E_i , we say that $(d_2, d_1; r^{-1})$ is an inverse edge in E_i ; and if $(d_1, d_2; r_1)$, $(d_2, d_3; r_2)$, ..., $(d_m, d_{m+1}; r_m)$ is a sequence of edges or inverse edges such that there is at most one occurrence (reversed or not) of each element in the sequence, then we say that $\langle r_1 r_2 \dots r_m \rangle$ is a relational path (or simply, path) from d_1 to d_{m+1} in the relational graph, and that d_{m+1} is reachable from d_1 by the label $\langle r_1 r_2 \dots r_m \rangle$.

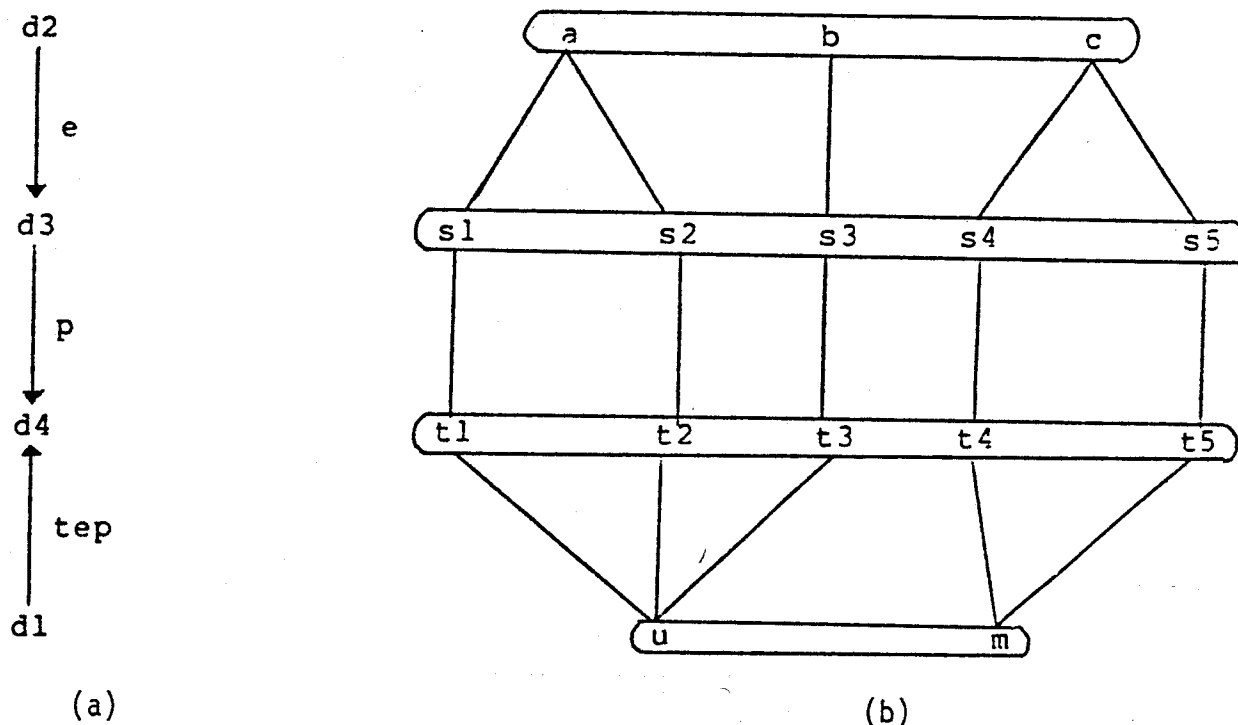


Fig. 2.2 (a) Natural relational graph for the abstract structure I_1 of Example 1, and (b) a description of the data interrelationships.

We can therefore interpret a query mapping as a function from relations in a conceptual model onto their shortest "equivalent" path in a relational graph, where the notion of equivalence is as for sequences of relations. Furthermore, a labelled reachability graph, i.e. a graph that represents such a path as a single labelled edge, represents the transitive closure of the relations of the abstract structure and therefore can be used to determine whether or not the graph implements a particular conceptual model.

Informally, a relational graph depicts the basic components and connections for a relational abstract structure, and thus characterizes

how a conceptual model is implemented: the vertices represent the domains and the labelled edges represent the explicit relations. Each edge can therefore be interpreted as representative of a two-column table of elements in which each column contains elements from one of the domains of the relation and each row denotes a relationship between two data values. Alternative graphs over the same set of vertices represent alternative choices for which tables to store, as long as the labelled reachability graph is identical to, or contains, the given relational graph.

The construction of $H_i = (V_i, L_i, E_i)$ follows in a natural way from $I_i = (D_i, R_i, \psi_i)$, thus we shall refer to H_i as the natural relational graph for I_i . However, there may be other relational graphs that can represent I_i . In the rest of the thesis, whenever a relational graph H and an abstract structure I are specified with the same subscript, it is implied that they are isomorphic images of each other.

Definition 3. A relational graph $H_j = (V_j, L_j, E_j)$ is said to support the relational graph $H_i = (V_i, L_i, E_i)$ if

1. there is a one-to-one correspondence between V_i and V_j
2. for all $r \in L_i$ there exists a path p in H_j such that $p \equiv r$.

Let $\hat{F}(H_i)$ denote the set of all the relational graphs supporting H_i , let $\hat{G}(I_i)$ denote the set of the corresponding abstract structures,

$\hat{G}(I_i) = \{I_j : H_j \in \hat{F}(H_i)\}$, and let $F(H^*)$ denote the set of the isomorphic images of the implementations of I , $F(H^*) = \{H_i : I_i \in G(I)\}$. In the following sections we will return to these concepts, and we will characterize the above sets in order to solve the efficiency problem, i.e. the problem of finding an efficient abstract structure implementing a given conceptual model.

2.1.3 Efficiency Parameters

The execution efficiency of abstract structures is typically a function of run time and storage space. For example, in (binary) relational databases, these two depend primarily on the number of tuples accessed to respond to a query and the total number of tuples stored, respectively.

In the following, time will be defined in terms of maximum and average response time in the given implementation for queries in I , and space will be defined in terms of the total number of relations stored in the implementation. In fact, the difference in size between relations may be very large, so an alternative measure of complexity could be to count the number of tuples, and consider its effect on the time required to join relations together. In fact, in Section 3.4 we refer to the number of couples of data elements when measuring the space complexity of particular abstract structures, but in general, in this research we simplify calculations by counting "virtual" relations of average size. Let us now formally introduce these parameters.

Given a query $q \in R$ and given $H_i \in F(H^*)$ (i.e., the isomorphic image of an abstract structure I_i implementing I), we define the retrieval time for q in H_i as

$$\Psi_i(q) = |\psi_i(q)|$$

that is, the retrieval time for a query q in H_i is the number of relations in I_i whose composition is needed to computer q . We will also define the maximum retrieval time as

$$\Psi(H_i) = \max_{q \in R} \Psi_i(q)$$

Definition 4. A relational graph $H_i \in F(H^*)$ is said to be d-time efficient if $\forall q \in R \forall H_j \in F(H^*)$ we have

$$\Psi_i(q) \leq \Psi_j(q) * d$$

Informally, every search in I_i takes at most d times more steps than the equivalent search in any other relational abstract structure implementing I , i.e. it is at most d times worse than the time-optimal structure.

Let us now consider the space complexity. Since we are considering relations of average size, then the parameter to evaluate the space complexity of I_i , also called the space requirement for I_i , is given by the total number of stored relations counting each relation and its inverse as only one relation. That is, the space requirement

$\Phi(H_i) = |R_i|/2$ is the number of arcs in $h(H_i)$. Let $h(H_i)$ be the skeleton of H_i , i.e., the undirected multigraph obtained from H_i by eliminating the orientation of the edges. Ignoring the labels, and considering each edge together with its reversed edge as one single arc.

Definition 5. A relational graph $H_i \in F(H^*)$ is said to be d-space efficient if for all $H_j \in F(H^*)$ we have

$$\phi(H_i) \leq \phi(H_j) * d$$

Informally, the space requirement for I_i takes at most d times more stored relations than any other relational abstract structure implementing I , i.e. it is at most d time worse than the space-optimal structure.

The choice of implementation, unfortunately, can seldom be made on the basis of time and space alone, but rather depends on both. We therefore define two further performance measures to account for both time and space together.

Given $H_i \in F(H^*)$, let $\text{space}(H_i)$ be the minimum ϵ such that H_i is ϵ -space efficient, and $\text{time}(H_i)$ be the minimum ϵ such that H_i is ϵ -time efficient.

Definition 6. A relational graph $H_i \in F(H^*)$ is d-time optimal if it is d-time efficient and $\forall H_j \in \{H_k \in F(H^*) \mid \text{time}(H_k) \leq d\}$
 $\text{space}(H_i) \leq \text{space}(H_j)$

Informally, H_i is d-time optimal if it is a d-time efficient graph with minimum space requirement.

Definition 7. A relational graph $H_i \in F(H^*)$ is d-space optimal if it is d-space efficient and $\forall H_j \in \{H_k \in F(H^*) \mid \text{space}(H_k) \leq d\}$

$$\text{time}(H_i) \leq \text{time}(H_j)$$

In other words, H_i is d-space optimal if it is a d-space efficient graph with minimum retrieval time over all the queries.

2.2 Fundamental Properties

In this section we will establish a fundamental equivalence between the general efficiency problem (i.e., the problem of finding an efficient abstract implementation of a conceptual model I) and the reduced efficiency problem (i.e., the problem of finding an efficient graph supporting a perfect implementation of I , cfr. Def. 3).

The contribution of this result is that, if I_i is a perfect implementation, then the reduced problem and the original problem are the same; that is, by finding a solution to the reduced problem we do not neglect any better solution to the general problem.

Therefore, the process to find an efficient abstract structure implementing a given conceptual model I can be reduced to the execution of the following basic steps:

Solution To The Efficiency Problem

(see Figure 2.3)

- step 1: find a perfect implementation I_i of I
- step 2: derive the natural graph H_i
- step 3: find in $\hat{F}(H_i)$ a graph H_j that minimizes certain costs
- step 4: derive the abstract structure I_j having H_j as natural graph

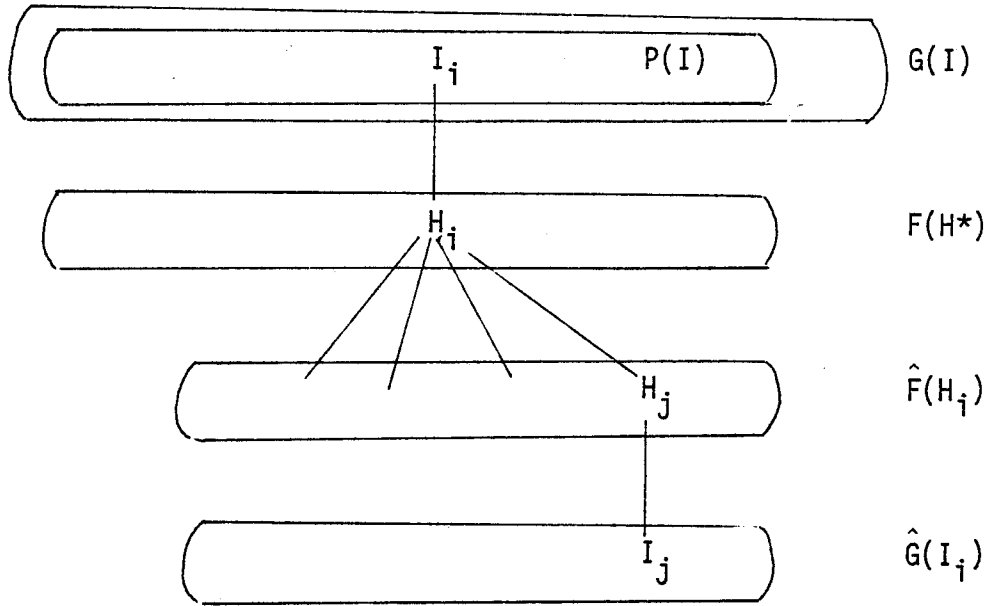


Fig. 2.3 Graphical representation of the steps to be executed to find an efficient abstract structure implementing the conceptual model I.

Because of the isomorphism between abstract structures and relational graphs, steps 2 and 4 can be easily performed. In Section 2.2.2 we will present a method to find a perfect implementation of I , thus solving step 1. Chapters 3 and 4 will treat the problem posed at step 3. Let us now analyze the properties of the sets $\hat{F}(H_i)$ and $\hat{G}(I_i)$, and derive the desired equivalence.

2.2.1 Equivalence Properties

Let us now prove some important properties of the sets $F(H^*)$ and $\hat{F}(H_i)$ when I_i is a perfect implementation, and discuss their relevance to the solution of the efficiency problem.

Property 2.1 Let $I_i, I_j \in P(I)$. Then $H_i \in \hat{F}(H_j)$.

proof: Since $I_i, I_j \in P(I)$, then $C(R_i) = C(R_j) = C(R)$. Therefore $V_i = V_j$. In order to complete the proof, we have to show that for all $r \in R_j$ there exists a path p in H_i such that $p \equiv r$. Given $r \in R_j$, obviously $r \in (R_j)^*$; since $I_i, I_j \in P(I)$, then $(R_j)^* = (R)^* = (R_i)^*$. Therefore $r \in (R_i)^*$, i.e., there is a path in H_i equivalent to r .

As a consequence, if I_i is a perfect implementation then $\hat{G}(I_i)$ contains all perfect implementations of I .

Corollary 2.1 If $I_i \in P(I)$ then $P(I) \subseteq \hat{G}(I_i)$.

Another interesting property of the set $\hat{F}(H_i)$ is given by the following

Property 2.2 Let $I_i, I_j \in P(I)$. Then $\hat{F}(H_i) = \hat{F}(H_j)$.

proof: Given a relational graph $H' = (V', L', E')$, we will prove that if $H' \in \hat{F}(H_i)$ then $H' \in \hat{F}(H_j)$ and vice versa. If $H' = H_i$ then, by Property 2.1, it follows that $H' \in \hat{F}(H_j)$. If this is not the case, let

$r \in R_j$. Since $H_i \in \hat{F}(H_j)$ then there is a path p in H_i such that $p = \langle r_1 r_2 \dots r_n \rangle \equiv r$, where $r_k \in R_i, 1 \leq k \leq n$. Since $H' \in \hat{F}(H_i)$, then for each $r_k, 1 \leq k \leq n$ there exists a path p_k such that $p_k = \langle r_{k-1} r_{k-2} \dots r_{k-k'} \rangle = r_k$, where $r_{k-s} \in L', 1 \leq s \leq k'$.

Let us consider the path $p' = \langle p_1 p_2 \dots p_n \rangle$ in H' . Obviously $p' \equiv p \equiv r$. Furthermore, since $H' \in \hat{F}(H_i)$ then there is a one-to-one correspondence between V' and V_i , and, since $H_i \in \hat{F}(H_j)$, there is a one-to-one correspondence between V_i and V_j . Therefore $H' \in \hat{F}(H_j)$ implies $H' \in \hat{F}(H_j)$. The converse follows in a similar way.

That is, all the sets of graphs supporting a perfect implementation contain exactly the same elements. We will now use this result to prove that the set of graphs supporting the isomorphic image of a perfect implementation of I is isomorphic to the isomorphic images of all the abstract structures implementing I .

Property 2.3 Let $I_i \in P(I)$. Then $G(I) = \hat{G}(I_i)$.

proof: Let $H_j = (V_j, L_j, E_j)$ be a relational graph. We will prove that if $H_j \in F(H^*)$ then $H_j \in \hat{F}(H_i)$ and vice versa.

(1) $H_j \in F(H^*) \implies H_j \in \hat{F}(H_i)$: we have to show that $V_j \iff V_i$ and that for all $r \in R_i$ there exists a path p in H_j equivalent to r . Since both H_i and H_j belong to $F(H^*)$, then $C(R_j) \iff C(R) \iff C(R_i)$, i.e., $V_j \iff V_i$. For each $r \in R_i$,

obviously $r \in (R_i)^*$. Since $I_i \in P(I)$, then $(R_i)^* = (R)^*$; therefore $r \in (R)^*$. From $H_j \in F(H^*)$ it follows that $(R_j)^* \supseteq (R)^*$. That is, $r \in (R_j)^*$.

(2) $H_j \in \hat{F}(H_i) \implies H_j \in F(H^*)$: we have to show that $D_j \supseteq D$, $(R_j)^* \supseteq (R)^*$ and that $C(R_j) \iff C(R)$. Since $H_j \in \hat{F}(H_i)$ and $R_i \in P(I)$, then $C(R_j) \iff C(R_i) \iff C(R)$ and $(R_j)^* \supseteq (R_i)^* = (R)^*$. Now $D_u \cup \{\delta\} \supseteq \{\delta\} \cup \bigcup_{d \in C(R_j)} d = \bigcup_{r \in R_j} \{x,y: (x,y) \in r\} \supseteq \bigcup_{r \in (R_i)^*} \{x,y: (x,y) \in r\} = D \cup \{\delta\}$.

Therefore $D_j \supseteq D$, and the proof is completed.

The above result gives us the key to attack the efficiency problem, i.e., the problem of choosing an efficient abstract structure implementing a given conceptual model. In fact, to find a solution to this problem, we need to explore the "solution space", i.e., the set $G(I)$. It is not difficult to see that, given a conceptual model we can construct an infinite number of abstract structures implementing it (a good analogy for $G(I)$ is the set of the Turing machines implementing a given function). That is, $G(I)$ is an infinite set, and therefore a brute force search is meaningless.

The contribution of Property 2.3 is that we can now restrict ourselves to the analysis of the reduced efficiency problem, i.e., the efficiency problem on the set $\hat{F}(H_i)$ instead of $F(H^*)$, where I_i implements I .

As the rest of this thesis will show, if I_i is a perfect implementation, then the reduced problem is "tractable"; that is, we can characterize the set, determine bounds and develop algorithms to solve the problem.

2.2.2 Constructing a Perfect Implementation

In this section we will introduce an algorithm to construct a perfect implementation of a given conceptual model I .

Intuitively, we will construct an abstract implementation by removing from H^* as many edges as possible while the resulting structure is still a perfect implementation of I .

Let us order the relations in R and number them from 1 to n . Let $\langle i_1 \dots i_k \rangle$ denote the subset of R composed of r_{i_1}, \dots, r_{i_k} , where $i_j \leq i_{j+1}$, $1 \leq j < k$, and $\langle \rangle$ denotes the empty set. Given $\langle i_1 \dots i_k \rangle$, we will denote by $I\langle i_1 \dots i_k \rangle$ the structure $(D, R\langle i_1 \dots i_k \rangle, \psi\langle i_1 \dots i_k \rangle)$ defined recursively as follows:

-- $R\langle \rangle = R$;

-- $\psi\langle \rangle = \mathcal{I}$;

-- $R\langle i_1 \dots i_k \rangle = R\langle i_1 \dots i_{k-1} \rangle - \{r_{i_k}, r_{i_k}^{-1}\}$; (notice that r_{i_k} and $r_{i_k}^{-1}$ are removed)

-- $\forall r \in r \ \psi\langle i_1 \dots i_k \rangle(r)$ is $\psi\langle i_1 \dots i_{k-1} \rangle(r)$ where all the occurrences of r_{i_k} have been substituted by a sequence $q \in S(R\langle i_1 \dots i_k \rangle)$ of minimum length such that $q \equiv r_{i_k}$; if no such sequence exists, then $\psi\langle i_1 \dots i_k \rangle$ is undefined.

That is, $I\langle i_1 \dots i_k \rangle$ is the structure obtained from I by deleting

relations r_{i_1}, \dots, r_{i_k} and their inverses, and adjusting (if possible) the query mapping. The recursive definition gives us an algorithmic approach to the construction of $I\langle i_1 \dots i_k \rangle$ once we have $I\langle i_1 \dots i_{k-1} \rangle$.

Let us now informally describe the algorithm. Function $\text{REPRES}(I^*)$ determine a perfect implementation of I , using backtracking, as follows:

1. Consider the graph obtained from H^* by deleting an edge $(x, y; r_j)$, $1 \leq j \leq \phi(H^*)$, and check if the corresponding isomorphic structure still implements I (i.e., if $\psi\langle j \rangle$ is defined). If this is the case, then we have $I\langle j \rangle \in P(I)$ (this will be proved formally below); in addition, $I\langle j \rangle$ has one explicitly stored relation less than I does.
2. In a recursive fashion, given $I\langle i_1 \dots i_k \rangle \in P(I)$, we check if $I\langle i_1 \dots i_k i_s \rangle$ still implements I , $i_k < i_s \leq \phi(H^*)$.
3. If, at any given time, we find that the structure under examination is a minimal perfect implementation, we stop the execution of the algorithm.
4. In order to examine each implementation once only, we use the total ordering of the relation indices: from structure $I\langle i_1 \dots i_k \rangle$, we can examine only the structures $I\langle i_1 \dots i_k i_s \rangle$ with $i_s > i_k$. It is easy to show that the latter structures can be accessed only from $I\langle i_1 \dots i_k \rangle$, and only if $I\langle i_1 \dots i_k \rangle \in P(I)$.

We will now introduce the algorithm and make use of the above recursive definition. Recall $I^* = (D, R, \mathcal{P})$, and that $H^* = (V, L, E)$ is its isomorphic image.

type structure = (set of data; set of relations; mapping):

procedure SEARCH ($\langle i_1 \dots i_k \rangle$: set; $\psi \langle i_1 \dots i_{k-1} \rangle$: mapping;

found: Boolean; I_0 : structure);

begin

found = FALSE;

construct $\psi \langle i_1 \dots i_k \rangle$;

if $\psi \langle i_1 \dots i_k \rangle$ is defined then

begin

if $\phi(H^*) - k < \phi(H_0)$ than $I_0 = I \langle i_1 \dots i_k \rangle$;

if ($\forall r \in R - \langle i_1 \dots i_k \rangle$ ($R - \langle i_1 \dots i_k \rangle - r$)* $\neq R$) then

found = TRUE;

else begin

$i = i_k + 1$;

while ($i \leq \phi(H^*)$ and found = FALSE) do

begin

SEARCH($\langle i_1 \dots i_k, i \rangle$, $\psi \langle i_1 \dots i_k \rangle$, found, I_0);

$i = i + 1$;

end;

end;

end;

end;

For sake of readability, the above procedure contains some redundant calculations; in fact, we do not need to perform the "while" loop for those relations r which satisfied the test $(R-\langle i_1 \dots i_k \rangle - r)^* \neq R$ previously in the controlling "iff" condition.

The function constructing a perfect implementation of I by using the above recursive function is the following:

```
type structure (set of data; sets of relations; mapping);
functions REPRES(I*:structure) : structure;
  begin
    structure I0;
    found = FALSE;
    I0 = I*;
    SEARCH(<>, ψ<>, found, I0);
  end;
I0;
```

In order to prove the correctness of the algorithm, we will first prove the following lemma.

Lemma 2.1 Let $I\langle i_1 \dots i_{k-1} \rangle \in P(I)$ and let $\psi\langle i_1 \dots i_k \rangle$ be defined.

Then $I\langle i_1 \dots i_k \rangle \in P(I)$.

proof: Since $I\langle i_1 \dots i_k \rangle$ has exactly the same domains and the same set of data as $I\langle i_1 \dots i_{k-1} \rangle$, then we have only to show that $(R\langle i_1 \dots i_k \rangle)^* = (R)^*$.

Since $I_{\langle i_1 \dots i_{k-1} \rangle} \in P(I)$, then $(R_{\langle i_1 \dots i_{k-1} \rangle})^* = (R)^*$; since $\psi_{\langle i_1 \dots i_k \rangle}$ is defined, then $(R_{\langle i_1 \dots i_{k-1} \rangle})^* = (R_{\langle i_1 \dots i_k \rangle})^*$.

Using the above Lemma as an induction step and using the relation $I_{\langle \rangle} = (D, R, \mathcal{S}) \in P(I)$ as inductive basis, it is easy to show that $\{I_0 \in P(I)\}$ is a time-invariant predicate everywhere-in procedure SEARCH. That is,

Property 2.4 $I_0 \in P(I)$

Finally, let us state a property of I_0 that will be used in the next section:

Lemma 2.2 Let $I_0 \leftarrow \text{--REPRES}(I^*)$. Then, no substructure of I_0 is a perfect implementation.

that is, even when I_0 is not a minimal perfect implementation as formally defined, it is still "minimal" in the sense that no substructure of I_0 is a perfect implementation. The proof is obvious.

2.2.3 Hierarchical Conceptual Models

In this section we will define and analyze the properties of a particular class of conceptual models, the hierarchical conceptual models, and investigate the properties of their implementations.

Definition 8. A conceptual model is said to be hierarchical iff for every cycle $\langle r_1 r_2 \dots r_k \rangle \exists r_j \in \{r_1, r_2, \dots, r_k\}$ such that

$$r_j = \langle r_{j+1} \dots r_k r_1 \dots r_{j-1} \rangle^{-1}$$

Because every relationship of interest is explicitly indicated in the conceptual model, if there are more than one relationships between a pair of domains, any two of them form a cycle. Thus, Definition 8 implies that in a hierarchical conceptual model there is at most one relationship (and its inverse) of interest between any two domains. We will state a simple method to determine if a given conceptual model is hierarchical later in this section.

If the conceptual model is hierarchical, then there is at least one corresponding minimal perfect implementation.

Property 2.5 If I is hierarchical then $\text{Basis}(I) \neq \emptyset$

proof: If I is hierarchical, then in any cycle there exists at least one relation equivalent to the inverse of the composition of the other relations in the cycle. Let us construct a new graph from H^* by

removing the edge associated with such relation, and apply this process recursively to the resulting graph. It is easy to observe that the isomorphic image of the final graph is a minimal perfect implementation of I .

Property 2.6 If I is hierarchical then $I_0 < \text{Basis}(I)$

proof: Let us first observe that if I is hierarchical, then $h(H_0)$ is a tree. In fact, if it is not a tree, then there is a cycle in H_0 , say $\langle r_1 \dots r_k \rangle$. Since I is hierarchical, then $r_j \in \{r_1, \dots, r_k\}$ such that $r_j = \langle r_{j+1} \dots r_k r_1 \dots r_{j-1} \rangle^{-1}$; therefore, the structure obtained by removing from I_0 the relation r_j would still be a perfect implementation, in contradiction to Lemma 2.2. Since $h(H_0)$ must be a tree, then the removal of any edge in H_0 will divide the graph into two disconnected subgraphs, and since $h(H^*)$ is connected then for all $r \in R_0 (R_0 - r)^* \neq R$.

The following property solves the problem of determining if a conceptual model is hierarchical or not.

Property 2.7 I is hierarchical if and only if $h(H_0)$ is a tree.

proof: In the proof of Property 2.6 we have shown that if I is hierarchical then $h(H_0)$ is a tree. Let us now prove the converse. If $h(H_0)$ is a tree, then there is only one path between any two given vertices; thus, all the relations in $(R_0)^*$ between two given vertices must be equivalent. From Property 2.4 and from the definition of perfect

implementation it follows that $(R_0)^* = (R)^*$. Therefore, also all the relations in $(R)^*$ (and thus in R) between two given domains must be equivalent, and this completes the proof.

In the rest of this section I is assumed to be hierarchical.

Because efficiency and optimality are judged relative to the performance of alternative implementations, the first task is to determine graphs which minimize each measure. We will now analyze some of these graphs and determine bounds on the efficiency. One of such graphs is H_0 , the isomorphic image of I_0 .

Property 2.8 H_0 is 1-space efficient

proof: Since I is hierarchical, then, by Property 2.7, $h(H_0)$ is a tree; that is, every graph requiring less space than H_0 is not connected and thus its isomorphic image does not implement I .

Using the above property, we can derive a closed formula to evaluate the space efficiency factor, $\text{space}(H_i)$, for an implementation $I_i \in G(I)$.

Lemma 2.3 Let $I_i \in G(I)$; then $\text{space}(H_i) = \lceil \phi(H_i)/(n-1) \rceil$

proof: Since H_0 is 1-space efficient (Property 2.8), then $\text{space}(H_i)$ is the only integer such that $(\text{space}(H_i)-1) \times \phi(H_0) < \phi(H_i) \leq \text{space}(H_i) \times \phi(H_0)$. Since I is hierarchical, then, from Property 2.7, it follows that $h(H_0)$

is a tree; that is $\phi(H_0) = n - 1$. Therefore, $\text{space}(H_i) = \lceil \phi(H_i)/(n-1) \rceil$.

The other limiting graph is H^* . Using the above Lemma, we can immediately determine its efficiency factor.

Property 2.9 H^* is $\lceil m/(n-1) \rceil$ -space efficient.

Let us observe that I^* contains explicitly all the relationships of interest in the application. Therefore, in general, H^* will be more similar to the complete graph than to H_0 ; that is, in general $m = O(n^2)$ and, thus $\text{space}(H^*) = O(n)$. The great advantage of H^* is obviously not with respect to space, but to time. In fact, in H^* each query requires the traversal of exactly one edge, i.e.,

$$\forall q \in R \quad |\mathcal{S}(q)| = 1. \quad \text{Thus}$$

Property 2.10 H^* is a 1-time efficient

Unlike for space, we are unable to derive a closed formula for the time efficiency factor of an arbitrary $I_i \in G(I)$; that is, we do not have an easy way, besides brute force, of checking the time efficiency of a structure. However, we can derive an upper bound for H_0 :

Property 2.11 H_0 is $(n-1)$ -time efficient

The proof follows directly from Property 2.7.

All the above results are summarized in Table 1.

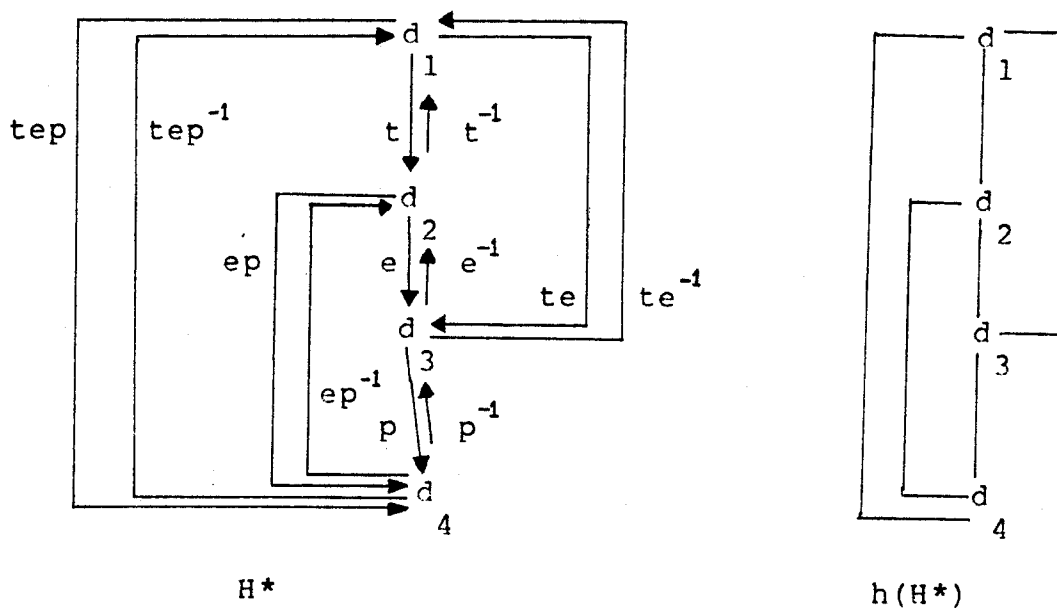
I_i	H_i	time (H_i)	space (H_i)
$I^*=(D,R,\mathcal{I})$	$H^*=(V,L,E)$	1	$\lceil m/(n-1) \rceil$
$I_0=(D,R_0,\psi_0)$	$H_0=(V,L_0,E_0)$	$n-1$	1

Table 1. Bounds on the efficiency of I^* and I_0 , where $n = |V| = |C(R)| = |C(R_0)|$ and m is the number of arcs in $h(H^*)$, i.e., $m = |R|/2$; usually $m = O(n^2)$.

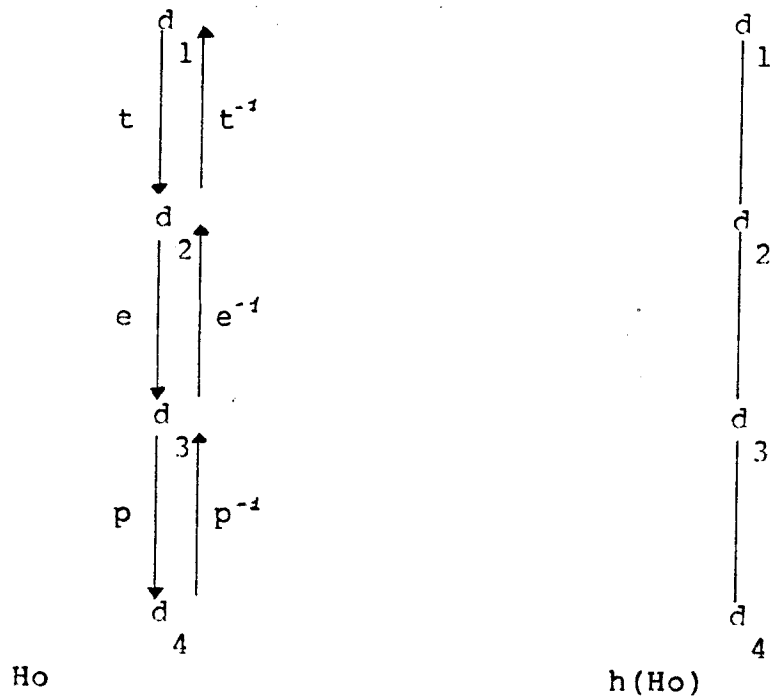
Let us illustrate the above properties by an example.

Example 2.

Consider the conceptual model of Example 1. The isomorphic image of $I^* = (D,R,\mathcal{I})$ is the graph $H^* = (V,L,E)$ shown below with its skeleton $h(H^*)$:



Let us construct the implementation I_0 from I^* by applying function $REPRES$, and let us assume that the indices of the relations in R are as follows: $tep = r_1$, $ep = r_2$, $te = r_3$, $t = r_4$, $e = r_5$ and $p = r_6$. Then, function $REPRES(I^*)$ will cause I_0 to be the abstract structure that was called I_3 in Example 1. The isomorphic image of I_0 and its skeleton are shown below:



Since $h(H_0)$ is a tree, then, by Property 2.7, it follows that I is hierarchical. We have $n = |V| = 4$ and $m = 6$. Therefore, by Property 2.9 it will be $space(H^*) = \lceil m/(n-1) \rceil = \lceil 6/3 \rceil = 2$; that is, H^* requires twice as much storage requirement as H_0 . On the other hand, by Property 2.11, $time(H_0) \leq n - 1 = 3$; that is, to answer a query in H_0 may take three times as much time as in H^* .

CHAPTER 3

STAR GRAPH APPROACH

3.1 INTRODUCTION

3.2 CONSTRUCTING THE STAR GRAPH

3.3 COMPLEXITY OF THE STAR GRAPH

3.4 CONSTRUCTING THE COMPLEMENT

CHAPTER 3

STAR GRAPH APPROACH

3.1 Introduction

In the previous sections, we have shown how to construct I_0 a perfect implementation of a given conceptual model I . In order to complete the solution of the efficiency problem, we must now determine an abstract structure that minimizes certain costs and whose image supports H_0 .

In this chapter, we consider hierarchical conceptual models. For such models we present a method to revise an abstract structure without altering its semantics; that is, the image of the revised structure still supports the original one. Under certain conditions which will be discussed later, it is possible, using this method, to reduce the retrieval time drastically with only a limited increase in space complexity. In fact, we show that, under certain conditions, the resulting structure is both space and time optimal and thus represents an optimal solution to the efficiency problem. In the rest of this chapter, we will present a method to revise I_0 , discuss the method's complexity, and show when the above-mentioned conditions occur.

Before proceeding, we will informally illustrate the basic concepts by an example.

Then we have

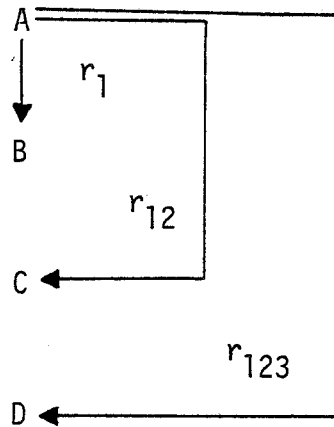
$$\Psi(H_0) = n$$

and, more importantly, assuming all queries equally likely, the average retrieval time is

$$\text{ave}_{q \in R} \Psi_0(q) = O(n)$$

That is, the average retrieval time is proportional to the number of relations.

Let us consider, instead, the following abstract structure $I+$:

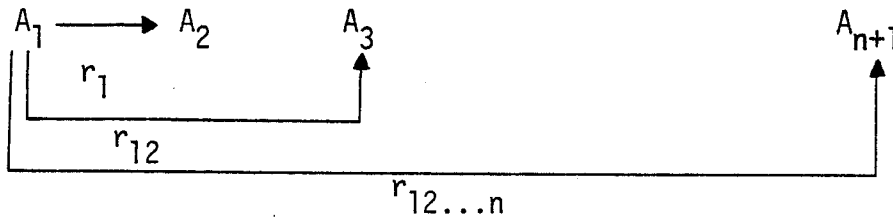


where $\psi+$ is as follows: $r_1 \rightarrow r_1$, $r_2 \rightarrow r_1^{-1} \cdot r_{12}$, $r_3 \rightarrow r_{12}^{-1} \cdot r_{123}$,
 $r_{12} \rightarrow r_{12}$, $r_{23} \rightarrow r_1^{-1} \cdot r_{123}$, $r_{123} \rightarrow r_{123}$.

It can be proven that $I+$ is a minimal perfect implementation of I . Let us now look at its maximum retrieval time

$$\Psi(H+) = \max_{q \in R} |\psi+(q)| = 2$$

The improvement over I_0 is not very significant, but let us consider the generalized situation:



In this case we still have

$$\Psi(H+) = 2$$

That is, the maximum (and thus obviously the average) retrieval time is independent of the number of relations.

The above example shows that the revised abstract structure $I+$ allows constant retrieval time over all the queries without any increase in space complexity. Therefore, $I+$ appears to be the desired solution to the efficiency problem. Because of its topological structure, we will refer to $H+$ as the star graph. Unfortunately, the revised structure $I+$ does not always implement I . In particular, when reconstructing a relation, say r_2 , in the above example, we were using several properties and equivalences of the relations. Namely, to reconstruct r_2 , we required that $r_2 = r_1^{-1} \cdot r_{12} = r_1^{-1} \cdot r_1 \cdot r_2$. In order for the

above equivalence to hold, it must be $r_1^{-1} \cdot r_1 = \iota$ (the identity relation), which was the case in our example. Unfortunately, this property ($r_1^{-1} \cdot r_1 = \iota$) is not generally true as can be seen from the example in figure 3.1.

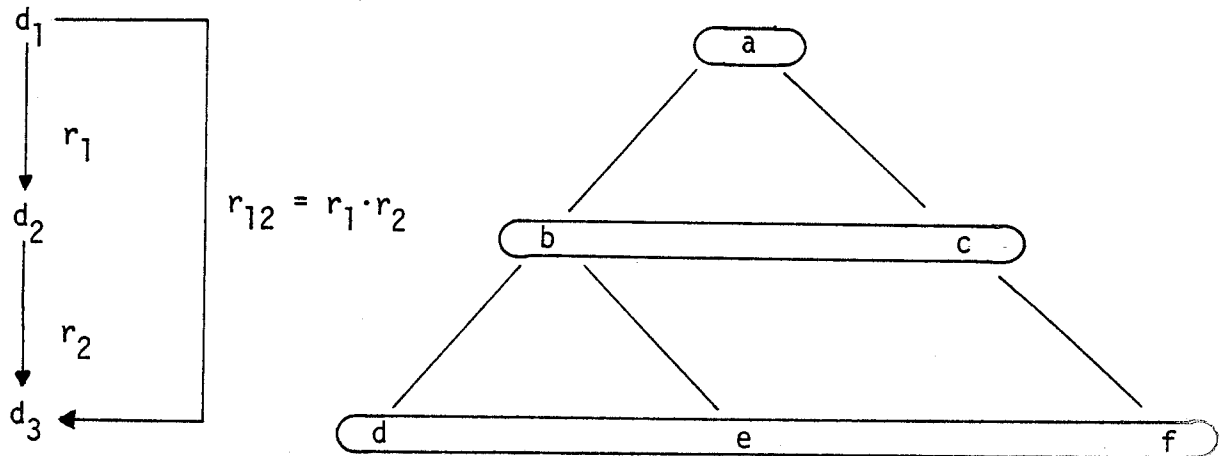


Figure 3.1 An example of problems with the inverse of a relation:

$$r_2^{-1} \cdot r_2 \neq \iota \text{ and } r_1^{-1} \cdot r_1 \neq \iota, \text{ but } r_1 \cdot r_1^{-1} = \iota \text{ and } r_2 \cdot r_2^{-1} = \iota \text{ where } r_1 = \{(a,b), (a,c), (\delta, \delta)\} \text{ and } r_2 = \{(b,d), (b,e), (c,f), (\delta, \delta)\} .$$

Note that $r_2^{-1} \cdot r_2 \neq \iota$ (e.g. $r_2^{-1} \cdot r_2[d] = \{d, e\}$ and that $r_1^{-1} \cdot r_1 \neq \iota$ (e.g. $r_1^{-1} \cdot r_1[b] = \{b, c\}$). What this really means is that the star graph shown in figure 3.2(a) does not implement I , because it is impossible to reconstruct r_2 from r_1 and r_{12} . This, however, does not mean that in our example we cannot construct a star graph implementing I . In fact, let us observe that $r_1 \cdot r_1^{-1} = \iota$ and that

$r_2 \cdot r_2^{-1} = 1$. This implies that in the star graph shown in figure 3.2(b) we can reconstruct r_1 ; in fact, $r_1 = r_{12} \cdot r_2^{-1} = r_1 \cdot r_2 \cdot r_2^{-1}$, but $r_2 \cdot r_2^{-1} = 1$ and therefore the above equivalence holds.

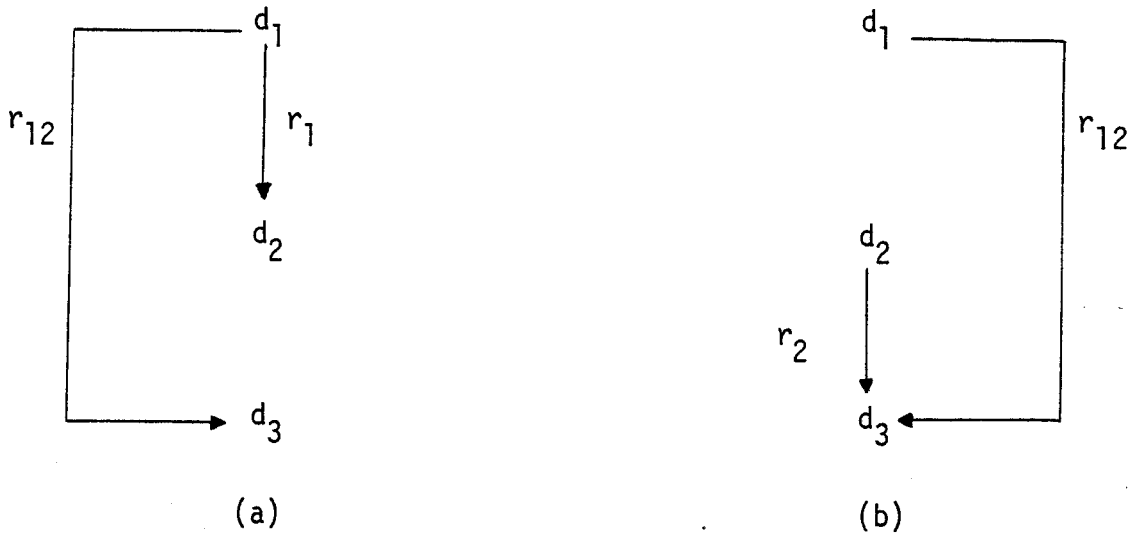


Figure 3.2 (a) Star graph whose isomorphic image does not implement the conceptual model of figure 3.1;
 (b) star graph that implements that conceptual model.

We will now formalize all these concepts by introducing the notion of the complement of a relation. Since I is hierarchical, then, for any two domains x and y , there is exactly one relation of interest between them, denoted r_{xy} .

Definition 9 The complement of a relation $r_{xy} \in R$ is the relation

$$\bar{r}_{xy} = [y,x] \text{ such that } \forall z \in C(R) \bar{r}_{xy} \cdot r_{xz} = r_{yz}$$

Informally, the complement \bar{r}_{xy} is that relation needed to proceed from y to x , such that the subsequent move from x to z answers the desired query. Let us anticipate that the complement of a given relation might not be in the set R ; we will come back on this point later in this chapter.

3.2 Constructing The Star Graph

We will now present an algorithm to construct the star graph I_+ from I_0 . The proposed algorithm uses the complement of a relation as defined in the preceding section.

type relational_graph = (set of vertices; set of labels; set of edges);

function STAR(H_0 :relational_graph):relational_graph;

 relational_graph H_+ ;

$V_+ := V_0$;

choose $x \in V_+$;

for all $y \in V_+$ and $y \neq x$ do

 begin

$L_+ := L_+ \cup r_{xy} \cup \bar{r}_{xy}$;

$E_+ := E_+ \cup (x,y;r_{xy}) \cup (y,x;\bar{r}_{xy})$;

end;

H_+ ;

Let us informally describe, using the terminology of relational graphs, how function STAR works. Given H_0 , we choose an arbitrary vertex $x \in V_0$; this vertex will become the center of the star graph H_+ that we are going to construct. From a topological point of view, H_+ will be composed of edges originating at the center. From a semantic point of view, we want H_+ still to implement the conceptual model, i.e., we want to be able to respond correctly to all the queries.

Since I is hierarchical, then given $y, z \in C(R)$, there is only one relation of interest between y and z , r_{yz} . We simulate in I^+ the relation r_{yz} by first going from y to x through \bar{r}_{xy} and then going from x to z through r_{xz} . The definition of complement guarantees that the result is exactly the desired one, i.e., $\bar{r}_{xy} \cdot r_{xz} = r_{yz}$. It is easy to see that to answer any query in I^+ , we need at most two steps; in fact, $r_{ab} \in R$,

$$|\psi^+(r_{ab})| = \begin{cases} 1 & \text{if } a = x \text{ or } b = x \\ 2 & \text{otherwise.} \end{cases}$$

This observation, together with Property 2.10, leads to the following:

Property 3.1 H^+ is 2-time efficient

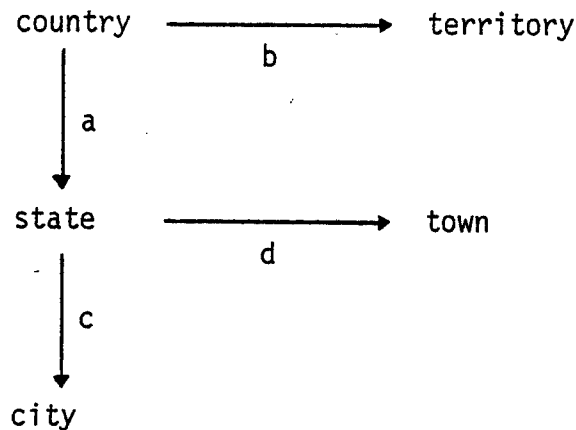
Evaluating the space complexity of the star graph is more difficult than evaluating its time complexity. In fact, the space requirement of H^+ depends on whether I^+ is a minimal perfect implementation of I or not. In the next section we will state a necessary and sufficient condition for $I^+ \in \text{Basis}(I)$ and analyze the corresponding space complexity. In Section 3.4 we will consider the case in which $I^+ \notin \text{Basis}(I)$, and we will analyze the increased space requirement of the star graph.

3.3 The complexity of the star graph

If for all $r \in R$ it happens $r^{-1} \cdot r = \iota$ (the identity relation), then $I+ \in \text{Basis}(I)$. This condition ($\{\forall r \in R r^{-1} \cdot r = \iota\}$) is a very strong (and unrealistic) one. However it is possible, in some conceptual model, that no all the relations satisfy the above condition but the star graph is still a minimal perfect implementation. An example is shown below.

Example 4.

Consider the following graph H_0 [Tsi2]



where the domains and the relationships are as follows:

country = $\{a_1, a_2, a_3\}$

state = $\{b_1, \dots, b_5\}$

territory = $\{c_1, \dots, c_6\}$

city = $\{d_1, \dots, d_7\}$

town = $\{f_1, \dots, f_7\}$

a = $\{(a_1, c_1), (a_1, c_2), (a_2, c_3), (a_3, c_4), (a_3, c_5), (a_3, c_6)\}$

b = $\{(a_1, b_1), (a_1, b_2), (a_2, b_3), (a_3, b_4), (a_3, b_5)\}$

c = $\{(b_1, d_1), (b_1, d_2), (b_1, d_3), (b_2, \delta), (b_3, d_4), (b_4, d_5),$
 $(b_4, d_6), (b_4, d_7), (b_5, \delta)\}$

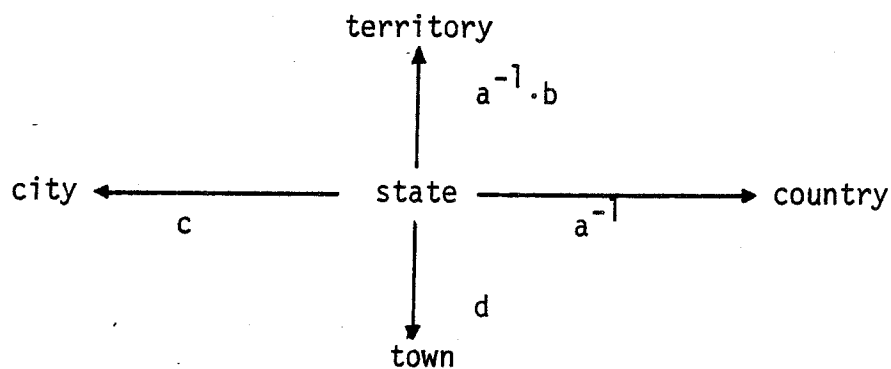
d = $\{(b_1, f_1), (b_2, f_2), (b_2, f_3), (b_3, f_4), (b_4, \delta), (b_5, f_5),$
 $(b_5, f_6), (b_5, f_7)\}$

and where the queries are

R = $\{a, b, c, d, b^{-1} \cdot a, a \cdot d, a \cdot c, d^{-1} \cdot c, b^{-1} \cdot a \cdot c, b^{-1} \cdot a \cdot d, \text{ and their inverses}\}$.

It is easy to observe that $a^{-1} \cdot a \neq i$, e.g. $a^{-1} \cdot a[c_5] = a[a_3] = \{c_4, c_5, c_6\}$.

At the same time, let us consider the following star graph obtained from H_0 :



It is trivial to show that $I+$ is a minimal perfect implementation of I even though not all the relations satisfy the above described property of the inverse.

By looking at the above example, we may find which conditions the relations must satisfy in order that $I+$ is a minimal perfect implementation of I . Let $x \in V+$ be the center of the star graph $H+$, and let $V(x) \subseteq R$ denote the set of relations between x and all the other domains. Then

Lemma 3.1 $I+ \in \text{Basis}(I)$ if and only if $\forall r \in V(x) \bar{r} = r^{-1}$.

proof: ($\forall r \in V(x) \bar{r} = r^{-1} \implies I+ \in \text{Basis}(I)$) $R+$ is composed of all the relations in $V(x)$ and their complements. Since $\forall r \in V(x) \bar{r} = r^{-1}$, then we can implement the complement of $r \in V(x)$ by using its inverse r^{-1} . Therefore, $R+$ will be composed of only the elements of $V(x)$ and their inverses. That is, $h(H+)$ is a tree. Therefore, for all $r \in R+ R \notin (R+ - \{r\})^*$.

($I+ \in \text{Basis}(I) \implies \forall r \in V(x) \bar{r} = r^{-1}$) If $I+ \in \text{Basis}(I)$ then all the complements must be elements of R . Since I is hierarchical, then there is only one relation between x and any domain y , r_{xy} , and only one relation between y and x , \bar{r}_{xy} . Therefore, the inverses must satisfy the defining equation of the complement, and the lemma is proved.

Let us now analyze the complexity of the star graph when $I+ \in \text{Basis}(I)$.

Property 3.2 Let $I+ \in \text{Basis}(I)$, then $H+$ is 1-space efficient

proof: From Lemma 3.1 it follows that if $I+ \in \text{Basis}(I)$ then

$\forall r \in V(x) \bar{r} = r^{-1}$; that is, $h(H+)$ is a tree or, equivalently, $\phi(H+) = n - 1$. Thus, by Lemma 2.3, it follows that $\text{space}(H+) = 1$.

By combining Properties 3.1 and 3.2 we can determine the time optimality of the star graph when I^+ is a minimal perfect implementation of I .

Property 3.2 Let $I^+ \in \text{Basis}(I)$, then H^+ is 1-space efficient

proof: From Lemma 3.1 it follows that if $I^+ \in \text{Basis}(I)$ then

$\forall r \in V(x) \bar{r} = r^{-1}$; that is, $h(H^+)$ is a tree or, equivalently, $\phi(H^+) = n - 1$. Thus, by Lemma 2.3, it follows that $\text{space}(H^+) = 1$.

By combining Properties 3.1 and 3.2 we can determine the time optimality of the star graph when I^+ is a minimal perfect implementation of I .

Property 3.3 If $I^+ \in \text{Basis}(I)$, then H^+ is 2-time optimal.

That is, if we want a retrieval time not greater than two, we cannot use less space than H^+ requires.

In order to establish the space optimality of I^+ , we need the following Lemma stated by Goldberg [Go1] and reported here without proof:

Lemma 3.2 If H_i is a strongly connected digraph without loops and with $|E_i|$ edges and $|V_i|$ vertices, and if H_i is not an elementary circuit then

$$d(H_i) \geq \lceil 2(|V_i| - 1) / (|E_i| - |V_i| + 1) \rceil$$

and this is the best possible result.

where $d(H_i)$ is the diameter of H_i .

Property 3.4 If $I+ \in \text{Basis}(I)$, then $H+$ is 1-space optimal.

proof: From Property 3.2 we know that $H+$ is 1-space efficient. Let $\hat{F}(H+)/1 = \{H_j \in \hat{F}(H+) \mid \text{space}(H_j)=1\}$ be the set of all the 1-space efficient structures implementing I . To complete the proof we have to show that $\forall H_j \in \hat{F}(H+)/1$ we have $\text{time}(H+) \leq \text{time}(H_j)$. Let us first observe that if $d(H+) \leq d(H_j)$ then $\text{time}(H+) \leq \text{time}(H_j)$. Since $H_j \in \hat{F}(H+)/1$, then $h(H_j)$ is a tree; that is, R_j contains exactly $2(n-1)$ relations. Therefore, by lemma 3.2, it follows that

$$d(H_j) \geq \lceil 2(|V_j|-1)/(|E_j|-|V_j|+1) \rceil = \lceil 2(n-1)/(2(n-1)-n+1) \rceil = \lceil 2(n-1)/(n-1) \rceil = 2. \text{ Since } d(H+) \leq 2, \text{ then the property holds.}$$

That is, with $n-1$ explicitly stored relations, we cannot have a better retrieval time than by using $I+$.

In conclusion, if $I+ \in \text{Basis}(I)$ (and because of lemma 3.1 this can be easily tested), then $I+$ is a solution to the efficiency problem. In fact, it allows constant retrieval time and requires only the minimum possible number of explicitly stored relationships.

3.4 Constructing the complement

In the previous section we have analyzed the complexity of the star graph when I^+ is a minimal perfect implementation, that is, when the inverses of the needed relations satisfy the defining equation of the complement. In this section we consider the case when this property does not hold, and we present an algorithm to construct the needed complements. The algorithm modifies the set of data elements by introducing "dummy" elements, thus increasing the space complexity of the star graph. After the presentation of the algorithm, we will analyze the degradation in space of the resulting structure I^+ , and present an upper bound on its space complexity.

The following function constructs the complement by modifying the set of data elements D . Some extra elements are introduced to augment certain inverse relations. These elements can be marked to disregard them when the inverse is used. While the time complexity of the new graph is unchanged, the space complexity is greatly affected by the number of new data elements, and thus by the number of couples of data elements, needed for storing the complemented relations.

```
type relation (set of couples of data);  
function COMPLEMENT(r:relation):relation;  
relation r;  
if  $r^{-1} \cdot r = i$  then  $\bar{r} := r^{-1}$ ;  
else
```

```
begin
   $\bar{r} := \emptyset$ ;
  for all  $a \in \text{ld}[r]$  do  $\bar{r} := \bar{r} \cup (\delta, a)$ ;
  for all  $a \in \text{rd}[r]$  do
    begin /* insert new element a' */
       $r := r \cup (a', \delta)$ ;
       $\bar{r} := \bar{r} \cup (a, a')$ ;
      for all  $s \in V(x)$  do
        if  $\exists q \in S(R)$  s.t.  $\psi_0(s) = \langle rq \rangle$  then
          for all  $(a, b) \in q$  do  $s := s \cup (a', b)$ ;
        end;
      end;
    end;
   $\bar{r}$ ;
```

We will illustrate how the function COMPLEMENT works through the following simple example. Consider the graph H_0 shown in Fig. 3.3(a), where the queries are $R = \{r_1, r_2, r_3, r_{12} = r_1 \cdot r_2, r_{23} = r_2 \cdot r_3, r_{123} = r_1 \cdot r_2 \cdot r_3,$ and their inverses}, and we are to construct the star graph H^+ having the vertex associated with domain B as center (Fig. 3.3(b)). The query mapping for the structure corresponding to H^+ is straightforward for all relations in R except r_3 . In order to reconstruct r_3 , we need to compose r_2 and r_{23} . To construct r_2 , the function COMPLEMENT performs the following operations. Consider the set $\text{rd}[r_2] = C = \{c_1, c_2, c_3\}$. For each $c_i \in C$ we create a new data

element c_i' which will augment the domain B , and we insert into r_2 the couple (c_i, c_i') . We now need to modify all the relations starting from B that use r_2 for the query mapping ψ_0 ; in this case, only r_{23} . In order to have $\bar{r}_2 \cdot r_{23} = r_3$, we insert into r_{23} the couples (c_i', d_j) where $d_j \in r_3[c_i]$. Because the domain B has been augmented, the relations r_1^{-1} and r_2 must also be augmented to include the new couples (c_i', δ) ; analogously \bar{r}_2 must include the couples (δ, b_i) . The resulting interrelationships for \bar{r}_2 and r_{23} are shown in Fig. 3.3(c).

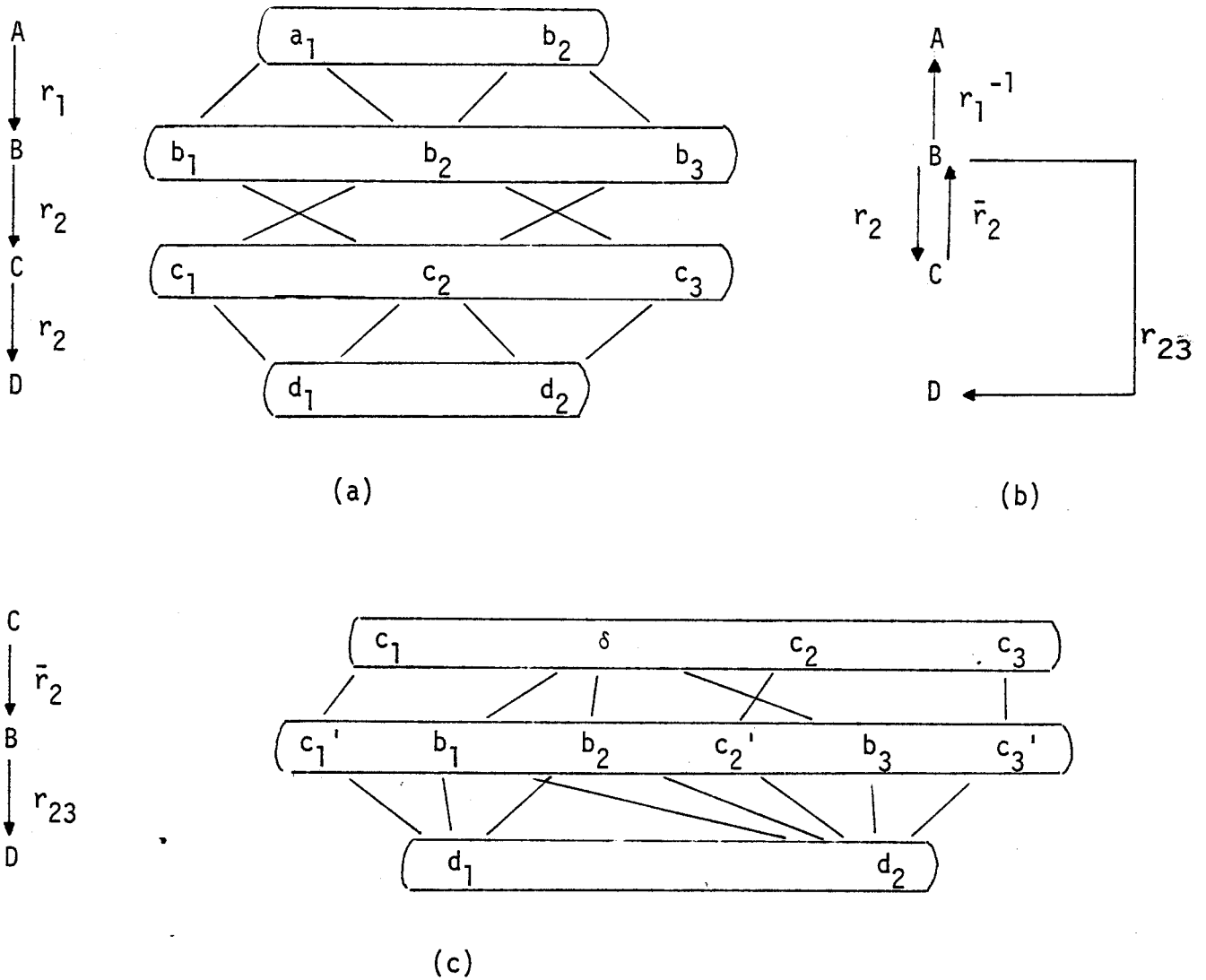


Fig. 3.3 (a) Natural relational graph H_0 and a description of the data interrelationships; (b) star graph H^+ ; (c) resulting data interrelationships for \bar{r}_2 and r_{23} .

Because we are modifying the domains without explicitly adding new relations, the number of relations cannot be used as a reliable measure of the space complexity; instead, the number of couples of data elements

needed to store the relations will give us a more accurate description of the space degradation. However, as will be shown later, the space degradation expressed in terms of couples of data elements can be seen as degradation equivalent to the introduction of new relations; therefore, we can still use the same notion of space requirement in order to describe the space complexity of this graph.

Let us now analyze the degradation in space of the star graph when constructing complements by the function COMPLEMENT. All the new elements are added only in one domain, precisely x . In fact, $\forall r \in V(x)$ we have either $ld[r] = x$ or $rd[r] = x$. Therefore, every time we modify $ld[r]$ (and we modify it when constructing r) we change all the relations in $V(x)$. Let us analyze in detail how many couples of data elements in the worst case are introduced when using function COMPLEMENT. When we use function COMPLEMENT, we practically duplicate in the domain x all the elements of the other domains. In fact, given $r_i \in V(x)$, COMPLEMENT(r_i) creates a new element a' for each element $a \in d_i = rd[r_i]$. Relation r_i will be composed of $|d_i|$ couples of data elements (a, a') ; given $r_j \in V(x)$, the number of new couples of data elements inserted into r_j when complementing r_i is exactly the number of couples in the unique (recall I is hierarchical) relation between d_i and d_j , r_{ij} . Therefore, when complementing all the relations in $V(x) = \{r_1, \dots, r_m\}$, the total degradation will be

$$\text{Degradation}(m) \leq \sum_{i=1}^m (|d_i| + \sum_{i < j \leq m} |r_{ij}|)$$

where the ordering on the indices in the inner summation is needed in order not to count the same relation twice.

Let us observe that the number of couples of data elements in the complete graph H^* is exactly

$$||R^*|| = \sum_{i=1}^m \sum_{i < j \leq m} |r_{ij}|$$

At the same time, the summation $||C(R)|| = \sum_i |d_i|$ gives exactly the total number of data elements (not necessarily distinct) in all the domains. That is, $\text{Degradation}(m) = ||C(R)|| + ||R^*||$. Since $||C(R)|| \leq ||R^*||$, then we can write

$$\text{Degradation}(m) \leq 2 ||R^*|| .$$

That is, the space degradation (in terms of new couples of data elements inserted) due to the construction of the complements, is equivalent to the creation of at most $2 |R^*|$ new relations. (Recall $|R^*|$ is the number of relations and $||R^*||$ is the number of tuples.) Therefore we can still express the space complexity of the star graph in terms of its space requirement as follows:

Property 3.5 If $I+ \notin \text{Basis}(I)$ then $\text{space}(I+) = O(n)$

In other words, if the isomorphic image of the star graph is not a

minimal perfect implementation, then its space complexity is of the same order as that of I^* .

Finally, let us recall that the above analysis is a worst case analysis; therefore the upper bound of property 3.5 is very far from the values found in many practical applications.

CHAPTER 4

REDUNDANT GRAPH APPROACH

4.1 INTRODUCTION

4.2 CONSTRUCTING AN EFFICIENT REDUNANT IMPLEMENTATION

4.2.1 General Case

4.2.2 Special Cases

CHAPTER 4

REDUNDANT GRAPH APPROACH

4.1 Introduction

In the previous section we have introduced a method to generate an abstract structure supporting a given hierarchical conceptual model, and we have analyzed the conditions under which the revised structure is both time and space optimal. If the resulting structure is not a minimal perfect implementation of the conceptual model, then the degradation in space can be as bad as the insertion of $O(n^2)$ new relations.

In this chapter we will discuss a different approach to the construction of an efficient abstract structure implementing a given hierarchical conceptual model. Let us assume that the star graph I^+ constructed by function $STAR(I^*)$ is not a minimal perfect implementation of I . In this case, the star graph might not be a practical solution to our problem. In fact, both I^* and I^+ allow an $O(1)$ retrieval time but may require $O(n^2)$ explicitly stored relations. That is, in both cases we are going to have a large space complexity in order to keep the time complexity to $O(1)$. In this case, we might well ask if there is a better way to guarantee a constant retrieval time; that is, if there exists a structure that implements I with $O(1)$ retrieval time but with $o(n^2)$ explicitly stored relations. In the following sections we will give a positive answer to this question. In fact, we will present a method to revise I_0

to form a structure that allows a constant retrieval time with only $O(n \log n)$ explicitly stored relations.

Given a hierarchical conceptual model I and the corresponding abstract structure I_0 , it is a trivial observation that if we add more explicitly stored relations (i.e., if we add redundancy) to I_0 , then the retrieval time for some queries may be reduced while the resulting structure will still implement I .

Definition 10. An abstract structure $I_i = (D_i, R_i, \psi_i)$ is said to be a redundant implementation of the conceptual model I

$$\text{iff } I_i \in P(I) \text{ and } R_0 \subset R_i \subset R$$

In other words, a redundant implementation of I is a structure having I_0 as a substructure, and its relations are elements of R . A simple example of a redundant implementation of I is the structure I^* .

By definition, the set of redundant implementations of I is composed of I_0 and all the structures obtained by adding to R_0 a subset of the relationships in $R^* - R_0$. Therefore we are interested in finding in this set a structure that needs $O(n^2)$ explicitly stored relations and that guarantees a constant retrieval time.

In the following, we will prove that, given a hierarchical conceptual model I , there always exists a redundant implementation I_ϵ of I such that $I(H_\epsilon) \leq (n-3)\log(n) + n-1$. Better bounds can be proved for special cases.

4.2 Constructing an efficient redundant implementation

4.2.1 General Case

Given a hierarchical conceptual model I , let us use REPRESENT (I^*) to construct the structure I_0 . Since I is hierarchical, then (by Property 2.7) $h(H_0)$ is a tree. Before showing how to construct the redundant graph H_ϵ , let us introduce some terminology.

Given an undirected tree $T = (N, A)$, where N is a set of nodes and $A \subseteq N \times N$ is a set of arcs, and given a node $x \in N$, the degree of x is the number of arcs incident on x and will be denoted by $g[x]$. Any node $x \in N$ will partition the tree T into the $g[x]$ subtrees, $T_1(x), T_2(x), \dots, T_{g[x]}(x)$ as shown in Figure 4.1. Let $t_i(x)$ be the number of nodes in $T_i(x)$, and, without loss of generality, let $t_1(x) \geq t_2(x) \geq \dots \geq t_{g[x]}(x)$.

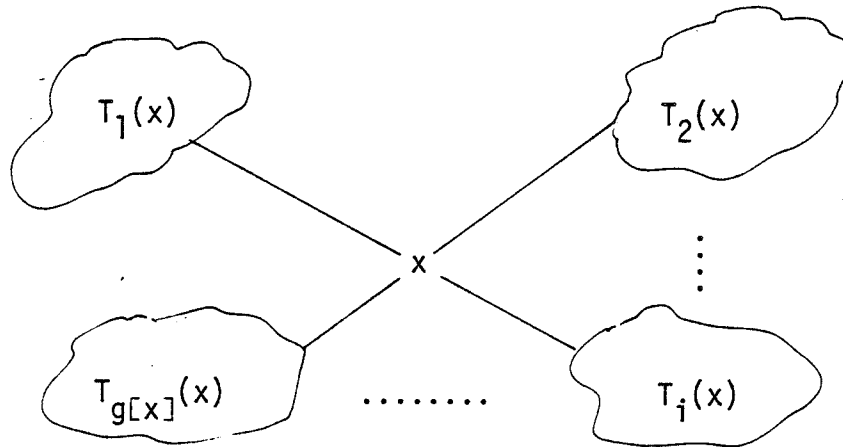


Figure 4.1 Partition of the tree T induced by the node $x \in N$

Let us now define the following measures:

$$\Delta_1(x) = t_1(x) - t_{g[x]}(x)$$

that is, $\Delta_1(x)$ represents the maximum imbalance in the number of nodes when partitioning the tree T into $g[x]$ subtrees; and

$$\Delta_2(x) = |t_1(x) - \sum_{j=2}^{g[x]} t_j(x)|$$

that is, $\Delta_2(x)$ is the imbalance in the number of nodes when partitioning T into two subtrees with $T_1(x)$ in one and all the other nodes (except x) in the other. Let

$$\Delta(x) = \min [\Delta_1(x), \Delta_2(x)]$$

be the imbalance factor for x . Now, a node $x_0 \in N$ is said to generate a best partition of T if $\forall x \in N \Delta(x_0) \leq \Delta(x)$.

Consider a procedure that creates an undirected graph T_ϵ from a tree T in the following way: find a node x_0 determining a best partition, connect all the nodes to x_0 , apply the same process recursively to all the subtrees rooted in x_0 . This may be coded as follows:

type graph = (set of nodes, set of arcs);

function REDUNDANT (T:graph):graph;

graph $T_\epsilon, T_{i\epsilon}$;

$T_\epsilon := T$;

find $x_0 \in N$ generating a best partition of T ;

for all $x \in N$ except x_0 and its direct children do


```
    Aε := Aε ∪ (x, x0);  
  for i=1 step 1 until g[x0] do  
    begin  
      Tiε := REDUNDANT(Ti(x0));  
      Aε := Aε ∪ Aiε;  
    end;  
Tε;
```

Using the above function, we can now construct the isomorphic image H_ϵ of the redundant implementation I_ϵ .

```
type graph = (set of nodes, set of arcs);  
type relational_graph = (set of vertices, set of labels, set of edges);  
function REDUNDANT_GRAPH (H0: relational_graph): relational_graph;  
  relational_graph Hε;  
  graph T0, Tε;  
  Hε := H0;  
  T0 := h(H0);  
  Tε := REDUNDANT (T0);  
  for all (x,y) ∈ Aε - A0 do  
    begin  
      Lε := Lε ∪ rxy ∪ ryx;  
      Eε := Eε ∪ (x,y;rxy) ∪ (y,x;ryx);  
    end;  
Hε;
```

We will now prove that the retrieval time in I_ϵ is at most two and that the space requirement is $O(n \log n)$.

Property 4.1 H_ϵ is 2-time efficient.

proof: We have to show that for any $r_{xy} \in R$, either $r_{xy} \in L_\epsilon$ or there exist in L_ϵ two relations r_{xz} and r_{zy} such that $\langle r_{xz} r_{zy} \rangle \equiv r_{xy}$. If $r_{xy} \in L_\epsilon$, then the property trivially holds. If this is not the case, consider the set $B(H_0)$ of the nodes in N_0 (and therefore vertices in V_0) chosen to generate best partitions when recursively applying REDUNDANT (T_0). Since $r_{xy} \notin L_\epsilon$, then $x \notin B(H_0)$ and $y \notin B(H_0)$; since $x, y \in N_0$, then there exists one and only one node $z \in B(H_0)$, generating a best partition during the recursive application, such that x and y belong to two different subtrees rooted in z . By construction, in T_ϵ there must be an arc from x to z and one from z to y . That is, there are in H_ϵ two edges, $(x, z; r_{xz})$ and $(z, y; r_{zy})$, such that $r_{xz} \cdot r_{zy} \equiv r_{xy}$ and the property is proved.

In order to determine the space complexity of H_ϵ , we need the following two lemmas.

Lemma 4.1 Let x_0 generate a best partition on T . Then

$$\lfloor \ln/g[x_0] \rfloor \leq t_1(x_0) \leq \lfloor \ln/2 \rfloor$$

proof: $t_1(x_0) \geq \lfloor \ln/g[x_0] \rfloor$ trivially follows from

$t_1(x_0) \geq t_2(x_0) \geq \dots \geq t_{g[x_0]}(x_0)$. Assume that $t_1(x_0) > \lfloor \ln/2 \rfloor$.

Then $t_1(x_0) = \lfloor \ln/2 \rfloor + \epsilon = \lceil (n-1)/2 \rceil + \epsilon$ for some $\epsilon \geq 1$, which implies that

$$\sum_{i=2}^{g[x_0]} t_i(x_0) = \lfloor (n-1)/2 \rfloor - \epsilon$$

Since $t_1(x_0) > \lfloor \ln/2 \rfloor$ then

$t_1(x_0) - t_{g[x_0]}(x_0) \geq t_1(x_0) - \sum_i t_i(x_0)$; thus

$$\Delta(x_0) = \Delta_2(x_0) = t_1(x_0) - \sum_{i=2}^{g[x_0]} t_i(x_0) = \lceil (n-1)/2 \rceil + \epsilon - \lfloor (n-1)/2 \rfloor + \epsilon$$

Let x' be the root of subtree $T_1(x_0)$. Node x' partitions T into $g[x']$ subtrees $T_1(x'), T_2(x'), \dots, T_{g[x']}(x')$ as shown in Figure 4.2.

Let $T_a(x')$ be the subtree composed of $T_2(x_0), \dots, T_{g[x_0]}(x_0)$ and node x_0 . We have

$$t_a(x') = \sum_{i=2}^{g[x_0]} t_i(x_0) + 1 = \lfloor (n-1)/2 \rfloor - \epsilon + 1$$

and

$$\sum_{j \neq a} t_j(x') = t_1(x_0) - 1 = \lceil (n-1)/2 \rceil + \epsilon - 1 .$$

$$\text{Now } \Delta(x') \leq \Delta_2(x') = \left| \sum_{j \neq a} t_j(x') - t_a(x') \right| =$$

$$= \left| \lceil (n-1)/2 \rceil + \epsilon - 1 - \lfloor (n-1)/2 \rfloor + \epsilon - 1 \right| = \Delta(x_0) - 2 .$$

That is $\Delta(x') < \Delta(x_0)$ which contradicts the hypothesis that x_0 is a best partition of T .

The above lemma generalizes a well known property for binary trees (see [Bre]).

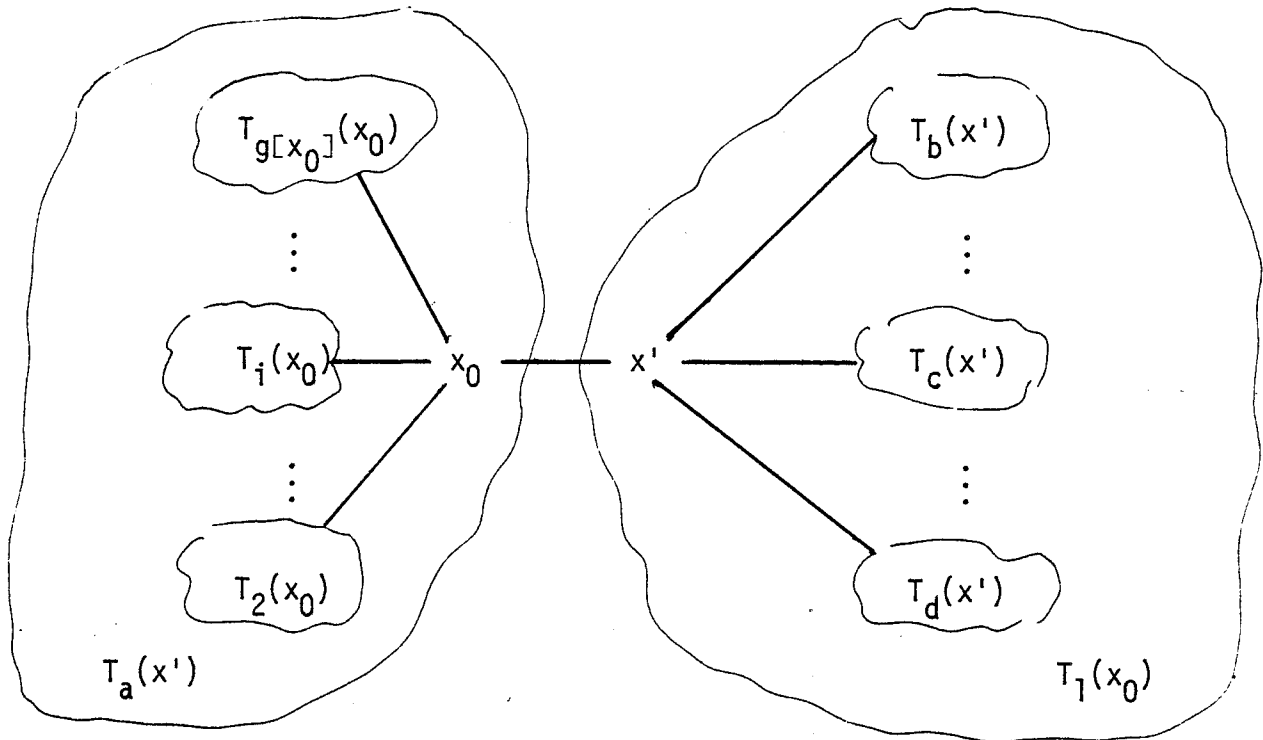


Fig. 4.2 Graphical description of Lemma 4.1.

Lemma 4.2 Let $A(n)$ be the number of arcs added by function REDUNDANT to a tree T with $n \geq 3$ nodes to form T_ϵ ; then

$$A(n) \leq (n-3) \log n$$

Proof: By induction. The lemma can be observed to hold for $n = 3$. Assume it to be true for $3 \leq n \leq m - 1$. The function REDUNDANT connects to x_0 all the nodes in T except x_0 and its children. It is then applied to $T_1(x_0), \dots, T_{g[x_0]}(x_0)$. Therefore

$$A(m) \leq n' + \sum_{i=1}^{g[x_0]} A(t_i(x_0))$$

where $n' = m - g[x_0] - 1$. For simplicity, let $t_i = t_i(x_0)$ and $g = g[x_0]$. Since for all i , $1 \leq i \leq g$, we have $t_i \leq m$, then (by inductive hypothesis) $A(t_i) \leq (t_i - 3) \log(t_i)$. Recall $\sum_i t_i = m - 1$.

Therefore

$$\begin{aligned} A(m) &\leq n' + \sum_i (t_i - 3) \log t_i \leq n' + (\sum_i (t_i - 3)) \log t_1 = \\ &n' + (m - 1 - 3g) \log t_1 \leq n' + n' \log t_1. \end{aligned}$$

Since $t_1 \leq \lfloor m/2 \rfloor$ by Lemma 4.1, then

$$\begin{aligned} A(m) &\leq n' + n' \log(\lfloor m/2 \rfloor) \leq n' + n' (\log(m) - 1) \leq n' \log m = \\ &(m - g - 1) \log m. \end{aligned}$$

Since $g \geq 2$, then $A(m) \leq (m - 3) \log m$ and the lemma is proved.

We can now determine the space complexity of the redundant implementation I_ϵ of I .

Lemma 4.3 $\phi(H_\epsilon) \leq (n - 3) \log n + n - 1$

The proof follows from Lemma 4.2 and by observing that $\phi(H_0) = n - 1$. Using this lemma together with lemma 2.3, we obtain

Property 4.2 $\text{space}(H_\epsilon) \leq \log n + 1$

Properties 4.1 and 4.2 establish a surprising fact shown in Table 2:

it is possible to reduce the order of magnitude of space in the complete graph without increasing the order or magnitude of time.

I_i	H_i	time(H_i)	space(H_i)
I^*	H^*	$O(1)$	$O(n^{**2})$
I_ϵ	H_ϵ	$O(1)$	$O(n \log n)$

Table 2 Worst case complexity of I^* and I_ϵ , where $n = |V|$ is the number of domains.

4.2.2 Special Cases

In the previous section we have stated a general upper bound on the space requirement of the redundant implementation I_ϵ . If we have some additional information about the topology of I_0 (e.g. $h(H_0)$ is a binary tree) then we can state tighter bounds on the space requirement.

In this section we examine three special cases; namely we consider the case when $h(H_0)$ is a strict k -ary tree, a perfectly balanced binary tree or a chain.

A strict k -ary tree is a tree where the degree of each vertex is either $k + 1$ or 1 .

Lemma 4.4 If $h(H_0)$ is a strict k -ary tree then

$$\phi(H_\epsilon) \leq (n-k-2) \log n + n - 1$$

The lemma follows from observing that in the proof of Lemma 4.2 we have $A(m) \leq (m-g-1)$ and that $g = \alpha[x_0] = k + 1$.

Corollary 4.1 If $h(H_0)$ is a strict binary tree, then

$$\phi(H_\epsilon) \leq (n-4) \log n + n - 1$$

Let us now consider the case of perfectly balanced binary trees, i.e., binary trees with $n = (2^{k+1}) - 1$ nodes and depth k .

Lemma 4.5 If $h(H_0)$ is a perfectly balanced binary tree then

$$\phi(H_\epsilon) \leq (n+1) \log(n+1) - 2n$$

Proof: Let $h(H_0)$ be a perfectly balanced binary tree. Then we construct $h(H_\epsilon)$ as follows: connect all the nodes to the root (except the direct children of the root) and then apply recursively the same process to the left and right subtrees. The total number of arcs added to form $h(H_\epsilon)$ is given by $A(n) = n - 3 + 2 A((n-1)/2)$. After $k = \log(n+1)$ iterations, the recurrence equation becomes

$$A(n) = \sum_{i=0}^{k-3} (2^{**i})((2^{**k-i})-1) - [3 \sum_{i=0}^{k-3} (2^{**i})] = k(2^{**k}) - 3((2^{**k})-1) + 1$$

that is, $A(n) = (n+1) \log(n+1) - 3n + 1$. Since $h(H_0)$ already contains $n - 1$ arcs, then the Lemma holds.

Another interesting tree is the chain, i.e., the tree in which each node has degree either one or two.

Lemma 4.6 If $h(H_0)$ is a chain then

$$\phi(H_\epsilon) \leq (n+1) \log(n+1) - 2n$$

Proof: In a chain with $n \geq 2$ nodes, there are only two leaves. The center of a chain is a node such that the difference of the distance of the node from the leaves is minimal. Let us construct $h(H_\epsilon)$ as follows: connect all the nodes to the center (except the direct children of the center) and apply recursively this process to the two subtrees rooted in the center. The number of arcs added to form $h(H_\epsilon)$ is given by

$A(n) = n - 3 + 2 A((n-1)/2)$, the same as for perfectly balanced binary trees. Therefore the above bound holds.

CHAPTER 5

GENERAL CONCEPTUAL MODELS

5.1 INTRODUCTION

5.2 HEURISTIC METHODS

5.3 RELATIONAL GRAPHS WITH ONE CIRCUIT

CHAPTER 5

GENERAL CONCEPTUAL MODELS

5.1 Introduction

In Chapters 3 and 4 we have been dealing with hierarchical conceptual models, and we have determined efficient ways to implement such models. If the conceptual model I is not hierarchical, i.e., $h(H_0)$ is not a tree, then the problem of finding an efficient abstract structure implementing I is much more difficult than in the hierarchical case. In fact, the properties proved in the previous chapters were based on the assumption that $h(H_0)$ was a tree, and thus there was a unique relationship (and its inverse) between any two domains. The presence of circuits in $h(H_0)$ greatly increases the number of possible different relationships between domains; for this reason a direct extension of the above proposed approaches is in many cases infeasible. However, if the topology of I_0 is quite simple, i.e., $h(H_0)$ contains only one circuit, then we can transform the structure in such a way that it becomes possible to apply one of the approaches for the hierarchical case; the resulting structure will have the same complexity as if the conceptual model were hierarchical.

In this chapter we will propose different methods to revise I_0 when the conceptual model I is not hierarchical, the particular method depending on whether there are more than one circuit or not. The methods for the general case are based on heuristics that use the star graph and

the redundant graph approach. Even though these heuristics do not allow us to determine direct bounds on the complexity (with the exception of the worst case), some experimental behaviour of the revised structures seems to show that they are quite useful tools to solve the average efficiency problem in the non-hierarchical case. If $h(H_0)$ contains only one circuit, then we are able to derive an algorithm (i.e., not merely heuristic) that is a direct extension of the hierarchical approach; the revised structures require the same time and space complexity as those for which I is hierarchical.

5.2 Heuristic Methods

In this section we present two heuristic methods based on the star graph and on the redundant graph approach to revise I_0 when $h(H_0)$ is not a tree. Let us informally describe the general idea behind these two heuristics. Both the star graph and the redundant graph approaches were designed to work with structures whose skeleton was a tree. If $h(H_0)$ contains circuits, then we can look at $h(H_0)$ as the union of edge-disjoint trees and apply one of the above methods to each of these trees.

In order to describe the proposed heuristics more precisely, let us again introduce some notation. Given an undirected graph $h(H_0)$, the skeleton of H_i , we will denote by $T[h(H_i)]$ a spanning tree of $h(H_i)$ and by $H/T[h(H_i)]$ the corresponding relational graph. Let us now present the function $HEU_STAR(H_0)$ that generates a relational graph $SH+ = (V, SL+, SE+)$. This function finds a spanning tree T of $h(H_0)$, generates a star graph on this tree, and applies the same process recursively to each connected component of the graph $T - h(H_0)$.

```
type relational_graph = (set of vertices, set of relations, set of edges);
type graph = (set of nodes, set of arcs);
function HEU_STAR( $H_0$ :relational_graph):relational_graph;
    relational_graph  $SH+, H_i, H_j, H_k$ ;
    graph  $h(SH+), h(H_i), T[h(H_i)]$ ;
     $SH++ H_i := H_0$ ;
```

```
while  $E_j \neq \emptyset$  do  
  begin  
    if  $h(H_i)$  is connected then  
      begin  
        construct  $T[h(H_i)]$ ; /* spanning tree */  
         $H_k := H/T[h(H_i)]$ ;  
         $H_j := \text{STAR}(H_k)$ ;  
         $SL+ := SL+ \cup L_j$ ;  
         $SE+ := SE+ \cup E_j$ ;  
      end;  
    else  
      begin  
        let  $H_1, \dots, H_p$  be the relational graphs corresponding  
        to the connected components;  
        for  $s = 1$  step 1 until  $p$  do  
          begin  
             $H_j := \text{HEU\_STAR}(H_s)$ ;  
             $SE+ := SE+ \cup E_j$ ;  
             $SL+ := SL+ \cup L_j$ ;  
          end;  
      end;  
  end;  
  
SH+;
```

Let us now analyze the complexity of the resulting relational graph

SH+ .

Property 5.1 If in each star graph constructed by function HEU_STAR the inverses of the relations to be complemented satisfy the defining equation of the complement, then

$$\phi(SH+) = \phi(H_0)$$

That is, we do not use more relations than H_0 ; the proof follows directly from Property 3.2.

The time complexity of SH+ is not as easy to derive. In fact, each query requires at most two steps to be answered if the corresponding path in SH+ remains within only one of the star graphs; that is, if the search is within one star graph, then the time complexity is the same as for the hierarchical case. If the path corresponding to a given query traverses more than one star graph component, then the retrieval time can be as bad as twice the number of traversed star graphs; that is, in the worst case the retrieval time can be twice the time required to answer the same query in H_0 as shown by the following example..

Example 5.

Let H_0 and SH+ be the graphs shown in Figure 5.1 and let relations a^{-1} , ab^{-1} , abc^{-1} , f^{-1} , fe^{-1} and fed^{-1} satisfy the defining equation of the complement. To answer the query $be = b \cdot e$ in graph SH+, we have to traverse the two star graphs components of SH+; in fact, $S_{\psi+}(be) = \langle a^{-1} ab f^{-1} ef \rangle$. Therefore, $S_{\psi+}(be) = 4$ while $\Psi_0(be) = 2$.

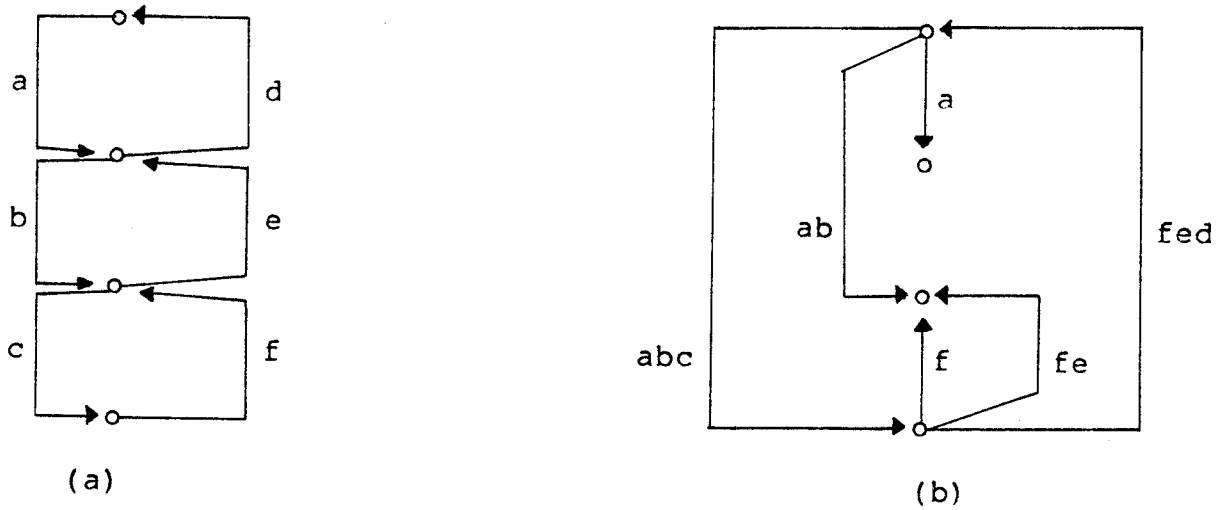


Fig. 5.1 (a) the relational graph H_0 and (b) the corresponding graph $SH+$ obtained through function HEU_STAR .

However, we may still obtain an improvement in performance by using $SH+$ instead of H_0 , as shown by the following example.

Example 6.

Let H_0 and $SH+$ be the graphs shown in Figure 5.2, and let b^{-1} , bc^{-1} and bcd^{-1} satisfy the defining equation of the complement. Then it is easy to observe that $\text{ave}_{q \in R} |\psi_0(q)| = 3$ while

$\text{ave}_{q \in R} |S\psi+(q)| = 2.6$ that is, to answer a query on $SH+$ requires on

the average almost half a step less than in H_0 .

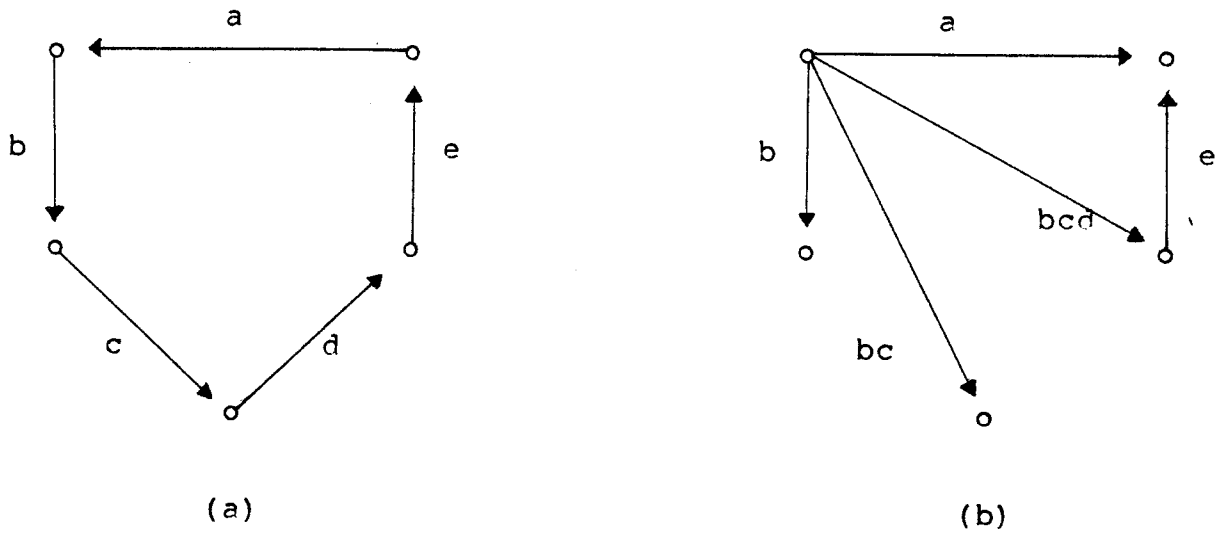


Fig. 5.2 Relational graph SH_+ (b) obtained by applying the heuristic function HEU_STAR to H_0 (a).

The second heuristic for constructing an implementation of a general conceptual model is based on the redundant graph approach. Similarly to the previous method, we partition the graph $h(H_0)$ into edge disjoint trees and then apply the redundant graph approach to each tree, generating the relational graph $SH_\epsilon = (V, SL_\epsilon, SE_\epsilon)$. The procedure is formally described below.

```

type relational_graph (set of vertices, set of relations, set of edges)
type graph = (set of nodes, set of arcs);
function HEU_RED( $H_0$ :relational_graph): relational_graph;
    graph  $h(H_i)$ ;
    relational_graph  $SH_\epsilon, H_i, H_j$ ;
     $SH_\epsilon := H_i := H_0$ ;
    while  $E_i \neq 0$  do

```

```
begin
  if  $h(H_i)$  is connected then
    begin
      construct  $T[h(H_i)]$ ;
       $H_j := \text{REDUNDANT\_GRAPH}(H/T[h(H_i)])$ ;
       $SE_\epsilon := SE_\epsilon \cup E_j$ ;
       $SL_\epsilon := SL_\epsilon \cup L_j$ ;
       $E_i := E_i - E_j$ ;
    end
  else
    begin
      let  $H_1, \dots, H_p$  be the relational graphs corresponding
      to the connected components;
      for  $s = 1$  step 1 until  $p$  do
        begin
           $H_j := \text{HEU\_RED}(H_s)$ ;
           $SE_\epsilon := SE_\epsilon \cup E_j$ ;
           $SL_\epsilon := SL_\epsilon \cup L_j$ ;
        end;
      end;
    end;
  end;
 $SH_\epsilon$  ;
```

Let us now analyze the space complexity of the new structure SI_ϵ .

Property 5.2 $\phi(SH_\epsilon) \leq \phi(H_0) \log \phi(H_0) + \phi(H_0)$

Proof: When constructing SH_ϵ , we decompose $h(H_0)$ into edge-disjoint trees and then construct the redundant graph for each tree. Therefore the total number of arcs added to form $h(SH_\epsilon)$ is given by

$$SA(n) = \sum_{i=1}^p A(n_i)$$

where n_1, n_2, \dots, n_p are the number of edges of the trees in which $h(H_0)$ has been decomposed, and $n = \phi(H_0)$. Therefore, by Lemma 4.2, we have $SA(n) \leq \sum_i (n_i - 3) \log n_i$. Since $\sum_i n_i = n$ then $\sum_i n_i \log n_i \leq n \log n$; that is, $SA(n) \leq n \log n$. Since $\phi(H_0) = n$ relations were already there, then the property holds.

In other words, the abstract structure constructed with the proposed heuristic has the same order of space complexity as the structure constructed with the redundant graph approach in the hierarchical case.

As for the time complexity, the observations made for the heuristic based on the star graph approach still hold. Again, if the query is expressed as a path within only one redundant subgraph (i.e., a subgraph constructed by applying the redundant graph approach to one of the trees), then the retrieval time is at most two. However, in the worst case it is proportional to the number of redundant subgraphs traversed by the path. Similarly to the other heuristic, if we consider the average behaviour then SH_ϵ may be more favourable than H_0 .

Example 7.

Let us consider the relational graph H_0 of Example 6 shown in Fig. 5.2. The relational graph generated by function $HEU_RED(H_0)$ is shown in Fig. 5.3. By simple calculation we obtain

$$\text{ave}_{q \in R} |\psi_0(q)| = 3 \quad \text{and} \quad \max_{q \in R} |\psi_0(q)| = 5$$

while

$$\text{ave}_{q \in R} |S\psi_\varepsilon(q)| = 2.2 \quad \text{and} \quad \max_{q \in R} |S\psi_\varepsilon(q)| = 3$$

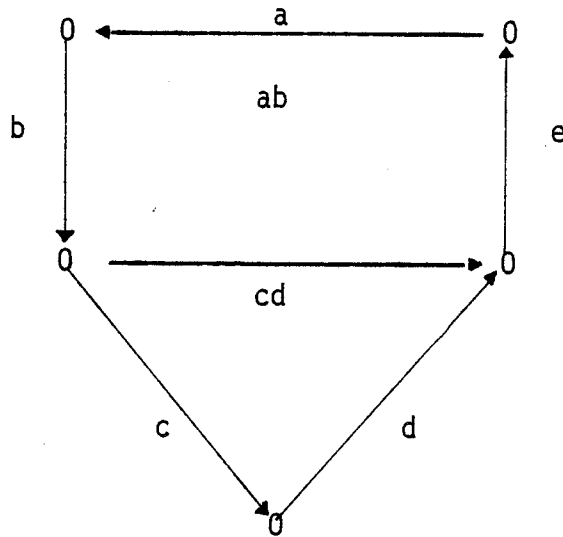


Fig. 5.3 Relational graph SH_ε generated by function $HEU_RED(H_0)$, where H_0 is the relational graph shown in Fig. 5.2(a).

5.3 Relational Graphs With One Circuit

Given a non-hierarchical conceptual model I , the skeleton $h(H_0)$ of the natural implementation I_0 is an undirected graph containing at least one circuit. If $h(H_0)$ contains exactly one circuit, then it is possible to reduce this case to the hierarchical one and thus apply the technique discussed in Chapter 4. In this section, we will present a direct method to revise I_0 whenever $h(H_0)$ contains exactly one circuit; the revised structure allows constant retrieval time and requires $O(n \log n)$ explicitly stored relations.

If $h(H_0)$ contains only one circuit, the number of arcs in $h(H_0)$ is exactly n , where n denotes, as usual, the number of nodes (i.e., domains). First let us informally describe how to revise H_0 , by means of a simple example.

Example 8.

Consider the relational graph H_0 shown in Fig. 5.4(a), where the queries are $R = R' \cup R''$, where $R' = \{r_{12}', r_{23}', r_{34}', r_{41}', r_{24}' = r_{23}' \cdot r_{34}', r_{31}' = r_{34}' \cdot r_{41}', r_{42}' = r_{41}' \cdot r_{12}', r_{21}' = r_{23}' \cdot r_{31}', r_{32}' = r_{31}' \cdot r_{12}', r_{43}' = r_{42}' \cdot r_{23}'\}$ are the "clockwise" relations, and $R'' = \{r_{ij}'' | r_{ij}'' = r_{ji}'^{-1}\}$ are the "counter-clockwise" relations. In other words, for any pair of domains x and y , there are two different queries (and their inverses) between x and y ; e.g. between x_2 and x_4 there are $r_{24}' = \langle r_{23}' r_{34}' \rangle$ and

$$r''_{24} = \langle r'_{12}{}^{-1} r'_{41}{}^{-1} \rangle \text{ with } r'_{24} \neq r''_{24} .$$

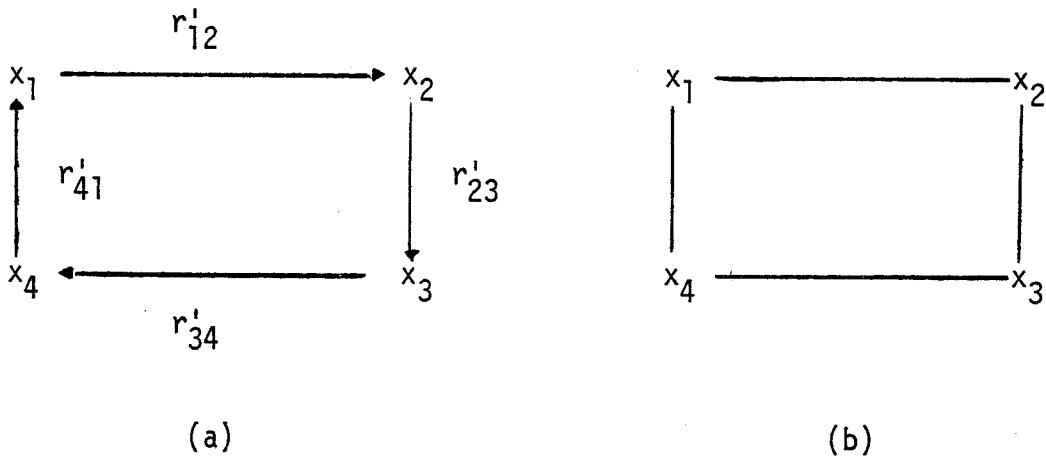


Fig. 5.4 (a) Relational graph H_0 ; (b) its skeleton $h(H_0)$ contains only one circuit.

The graph $h(H_0)$ can be transformed into a tree with $n + 1$ nodes by simply "splitting" an arbitrary node. That is, we choose an arbitrary node (e.g. x_1 in Fig. 5.4) and consider it to be two distinct nodes (x' and x'') . In this way we obtain the tree $h(\text{TH}_0)$ shown in Fig. 5.5(a) with the corresponding relational graph TH_0 .

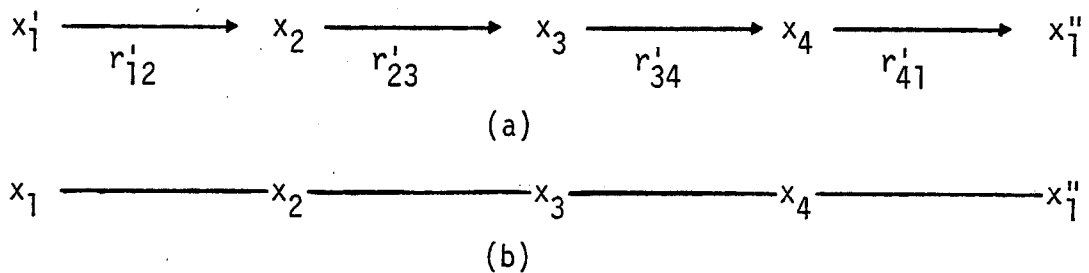


Fig. 5.5(a) Relational graph TH_0 obtained after "splitting" x_1 into x'_1 and x''_1 ; (b) the skeleton $h(\text{TH}_0)$ is a tree.

The next step is to apply the redundant graph approach to the structure TH_0 using only clockwise relations and their inverses. The resulting relational graph TH_ϵ is shown in Fig. 5.6.

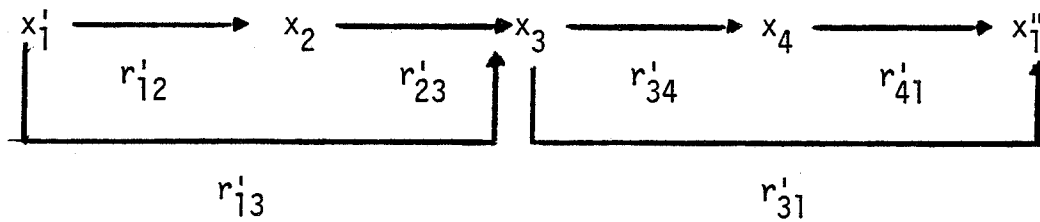


Fig. 5.6 Relational graph TH_ϵ obtained by the application of the redundant graph approach to TH_0 .

At this point, all the queries that do not traverse x_1 can be answered in at most two steps. In order to allow all the queries to have retrieval time at most two, all vertices x_i must be connected to x_1 by relation r_{i1}' ; and x_1 must be connected to all x_i by r_{1i}' . The final relational graph HH_ϵ is shown in Fig. 5.7.

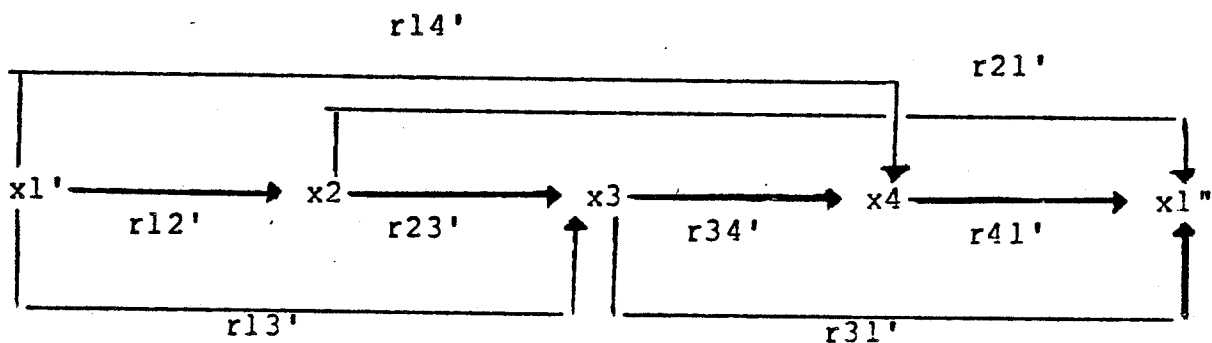


Fig. 5.7 Final relational graph HH_ϵ .

Let us now ask how many relations have been added. In the first step

("splitting" x_1) we did not add any relation. In the second step we have applied the redundant graph approach to a tree with 5 nodes. Therefore (Lemma 4.2) at most $2 \log 5$ new relations have been introduced. In the final step, we have connected twice to x_1 (once as x_1' , the other time as x_1'') all the vertices not already connected with both r'_{1i} and r'_{i1} . In the example, two new relations have been added, bringing the total number of relations added to R_0 to $A(4) = 4 \lfloor 2 \log 5 \rfloor + 6$.

That is $\phi(HH_\epsilon) = O(n \log n)$: the space complexity of the new structure is of the same order as the one obtained if the conceptual model were hierarchical.

Let us now extend these results to an arbitrary graph with one cycle, as shown in Fig. 5.8, where T_i denotes a tree rooted in node x_i .

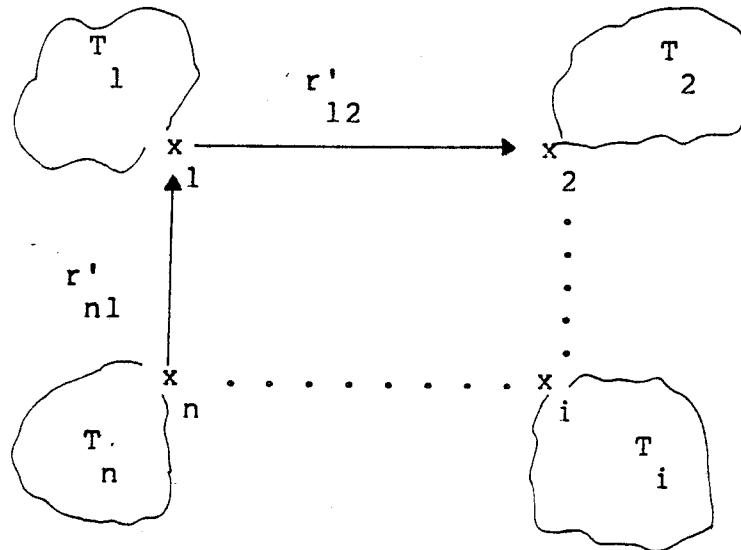


Fig. 5.8 Relational graph H_0 whose skeleton contains only one circuit.

The process to generate HH_ϵ from H_0 follows:

- step 1 choose an arbitrary vertex x_1 in the circuit, choose an arbitrary direction in the circuit and call that direction "clockwise"; transform the graph $h(H_0)$ into a tree $h(T_{H_0})$

by "splitting" node x_1 into x_1' and x_1'' .

step 2 apply the redundant graph technique to TH_0 using only clockwise relations and their inverses.

step 3 connect all vertices x_i to x_1 with r_{i1}' ; connect x_1 to all vertices x_i with r_{1i}' .

Property 5.3 $\phi(HH_\epsilon) \leq (n-2)\log(n+1)+3n-2$

Proof: In step 1 we do not add any relation. In step 2 we apply the redundant graph approach to a tree with $n+1$ nodes. Therefore, by Lemma 4.2, at most $(n-2)\log(n+1)$ new relations have been introduced. In the final step we doubly connect each node x_i to x_1 (if not already connected by r_{1i}' and r_{i1}'). That is, at most $2(n-1)$ new relations have been added. Therefore the total number of relations added to R_0 to generate HR_ϵ is $A(n) \leq (n-2)\log(n+1)+2(n-1)$. Since $\phi(H_0) = n$ relations were already there, then the bound holds.

Property 5.4 HH_ϵ is 2-time efficient

Proof: Let us consider four possible cases depending on the possible values of a given query $q \in R$.

Case 1 ($q=r_{1i}'$ or $q=r_{i1}'$ for some i): every such query requires only one step in HH_ϵ (see Step 3).

Case 2 ($q=r_{1i}''$ or $q=r_{i1}''$): since $r_{1i}'' = r_{i1}'$ and $r_{i1}'' = r_{1i}'$ then it will again take only one step to answer this kind of query.

Case 3 ($q=r'_{ij}$ or $q=r''_{ij}$ and q does not traverse x_1): because of the redundant graph approach used to construct TH_ϵ , then it will take at most two steps to answer this kind of query.

Case 4 ($q=r'_{ij}$ or $q=r''_{ij}$ and q traverses x_1): it will take exactly two steps; in fact, $r'_{ij} = \langle r'_{ij} r'_{ij} \rangle$ and $r''_{ij} = r'_{ji}^{-1} = \langle r'_{ji} r'_{ji}^{-1} \rangle = \langle r'_{ji}^{-1} r'_{ji}^{-1} \rangle$.

CHAPTER 6

CONCLUSIONS AND APPLICATIONS

6.1 SUMMARY

6.2 A FINAL EXAMPLE

6.3 OTHER APPLICATIONS

6.3.1 Picture Processing

6.3.2 Computer Networks

6.4 CONCLUSIONS AND EXTENSIONS

CHAPTER 6

CONCLUSIONS AND APPLICATIONS

6.1 Summary

Given a conceptual model, we can construct several different abstract structures implementing the model; that is, all the properties (relations among data) contained in the model are still available in the structure, either directly or through computation. Since the abstract structure contains the notion of "computation", we can introduce some abstract complexity measures that enable us to analyze the efficiency of a given structure and to compare different structures implementing a given conceptual model in order to determine which implementation is more efficient. In Chapter 2 we have introduced some abstract complexity measures and developed some tools to analyze the efficiency of the implementations of a given model. Let us now first briefly summarize the results obtained in Chapters 3 and 4, where the conceptual model was assumed to be hierarchical.

The first observation to be made is that given a conceptual model I , there is always a structure I^* implementing I . In the design of binary relational data bases, the structure $I^* = (D, R^*, \mathcal{F})$ is the description of the given problem. This structure has the following complexity (Properties 2.9 and 2.10):

space complexity	$\phi(I^*) = O(n^{**2})$
time complexity	$\psi(I^*) = O(1)$

where n is the number of domains in the model. In other words, I^* allows constant retrieval time at the expense of a quadratic space requirement.

Through function $\text{REPRES}(I^*)$ we can construct another implementation of I , I_0 , with the following complexity (Properties 2.8 and 2.11):

$$\begin{array}{ll} \text{space complexity} & \phi(I_0) = O(n) \\ \text{time complexity} & \psi(I_0) = O(n) \end{array}$$

that is, the order of the space complexity has been reduced at the expense of an increase in the order of the time complexity. Recall that in practical applications I_0 is the first (and often the only) implementation considered; in fact I^* is regarded as a description of the problem rather than as an implementation.

Through the redundant graph approach, we have been able to determine an alternative structure I_ϵ whose complexity is a compromise between the ones of I^* and I_0 (Properties 4.1 and 4.2):

$$\begin{array}{ll} \text{space complexity} & \phi(I_\epsilon) = O(n \log n) \\ \text{time complexity} & \psi(I_\epsilon) = O(1) \end{array}$$

The above structure is obviously more efficient than I^* ; in fact, it requires less space to achieve the same time complexity. A comparison between I_ϵ and I_0 involves a trade-off between time and space. However, in a paged environment (where the number of relations represents

the number of pages and the retrieval time gives the number of page-in/page-out operations) the trade-off is between having $O(n)$ pages and $O(n)$ page-in/page-out operations in the worst case to we process a query, or rather to have $O(n \log n)$ pages but a constant number of page-in/page-out operations.

The most interesting implementation is the one obtained through the star graph approach. In fact, if the resulting structure I^+ is a minimal perfect implementation, then (Properties 3.1 and 3.2)

space complexity	$\phi(I^+) = O(n)$
time complexity	$\psi(I^+) = O(1)$

that is, we can implement the conceptual model with only $O(n)$ relations (pages) and this structure allows constant retrieval time. Furthermore, we cannot do better (Property 3.3 and 3.4), that is, there does not exist any structure implementing I that needs less relations, and with $n - 1$ relations we cannot achieve better retrieval time. If I^+ is not a minimal perfect implementation than (property 3.5) its worst case complexity is comparable to that of I^* . These results are summarized in Table 3.

I_i	$\psi(H_i)$	$\phi(H_i)$
I^*	$O(1)$	$O(n^{**2})$
$I+\epsilon$ Basis(I)	$O(1)$	$O(n^{**2})$
$I_\&$	$O(1)$	$O(n \log n)$
$I+\notin$ Basis(I)	$O(1)$	$O(n)$
I_0	$O(n)$	$O(n)$

Table 3. Complexity of the implementations I^* , I_0 , $I+$ and I_ϵ described in Chapters 2 through 4.

Chapter 5 considered non-hierarchical conceptual models. If the structure I_0 is "simple" (i.e., $h(H_0)$ contains only one circuit) then we can apply a generalization of the redundant graph approach and construct an implementation II_ϵ that has the same properties (to within a constant) as the redundant implementation I_ϵ for the hierarchical case. These results are shown in Table 4.

I_i	$\psi(H_i)$	$\phi(H_i)$
I^*	$O(1)$	$O(n^{**2})$
II_ϵ	$O(1)$	$O(n \log n)$
I_0	$O(n)$	$O(n)$

Table 4. Time and space complexity of the implementation of a "simple" general conceptual model.

If this is not the case, then heuristic methods based on the star graph and on the redundant graph approach have been proposed to construct implementations of the given model. These abstract structures have the same space complexity as the corresponding structures for the hierarchical case. No direct bounds on the time complexity have been derived but the observed behavior of the retrieval time is lower than the one observed in I_0 .

6.2 A Final Example

In this section we will show how the formalism introduced in Chapter 2 can be used to describe a data base containing information about television companies in the world; the abstract structures I_0 and II_g implementing the corresponding model will be determined and their efficiency measured.

The problem under consideration is a list of information about world television companies taken from [Fr]; a sample of the data concerning the Canadian television companies may be found in Appendix C.

Let us first re-express the problem as a conceptual model. The set D of data is composed of all the elements in the domains defined by the relations of interest (possible queries); that is

$$D = \bigcup_{d \in (D)} d$$

where

$C(D) = \text{Nation} \cup \text{Name} \cup \text{Address} \cup \text{City} \cup \text{Telex} \cup \text{Cable} \cup \text{Tellephone}$
 $\text{Call} \cup \text{Cha} \cup \text{Kw}$, and the domains are:

Nation : set of names of nations;

Name : set of names of TV companies;

Address : set of addresses;

City : set of names of cities;

Telex : set of telex #;

Cable : set of cable #;

Telephone : set of telephone #;
Call : set of names of TV stations;
Cha : set of channel #;
Kw : set of Kilowatts.

The set R is composed of the queries that can be asked in the model; that is, R contains the following relations (for brevity, we will use in the following discussion only the underlined portions of the relations' name):

$[\underline{n}] = (\text{Nation}, \text{Name})$

e.g. $n[\text{Canada}] = \{\text{CBC}, \text{CTV}, \text{TVA}, \text{ORTQ}, \text{OECA}\}$

$[\underline{c}] = (\text{Name}, \text{Cable})$

$[\underline{t}] = (\text{Name}, \text{Telex})$

$[\underline{te}] = (\text{Name}, \text{Telephone})$

$[\underline{s}] = (\text{Name}, \text{Call})$

e.g. $s[\text{ORTQ}] = \{\text{CIVM}, \text{CIVQ}, \text{CIVO}\}$

$[\underline{w}] = (\text{Call}, \text{Address})$

$[\underline{i}] = (\text{Address}, \text{City})$

$[\underline{h}] = (\text{Name}, \text{Address})$

$[\underline{ch}] = (\text{Call}, \text{Cha})$

e.g. $ch[\text{CBIT}] = \{5\}$

e.g. $ch^{-1}[5] = \{\text{CBYT}, \text{CBIT}, \text{CBLT}, \text{CBXT}\}$

$[\underline{wi}] = w.i. = (\text{Call}, \text{City})$

$[\underline{txt}] = tx^{-1}.te = (\text{Telex}, \text{Telephone})$

$[\underline{txc}] = tc^{-1}.ca = (\text{Telex}, \text{Cable})$

[tec] = $te^{-1}.ca$ = (Telephone, Cable)

[chk] = $ch^{-1}.k$ = (Cha, Kw)

[chwi] = $ch^{-1}.w.i$ = (Cha, City)

[chw] = $ch^{-1}.w$ = (Cha, Address)

[kw] = $k^{-1}.w$ = (Kw, Address)

[kwi] = $k^{-1}.w.i$ = (Kw, City)

[nt] = $n.te$ = (Nation, Telephone)

[nx] = $n.tx$ = (Nation, Telex)

[nc] = $n.ca$ = (Nation, Cable)

[cas] = $ca^{-1}.s$ = (Cable, Call)

[txs] = $tx^{-1}.s$ = (Telex, Call)

[tes] = $te^{-1}.s$ = (Telephone, Call)

[nswi] = $n.s.w.i$ = (Nation, City)

[ns] = $n.s$ = (Nation, Call)

[swi] = $s.w.i$ = (Name, City)

[sk] = $s.k$ = (Name, Kw)

[sch] = $s.ch$ = (Name, Cha)

[sw] = $s.w$ = (Name, Address)

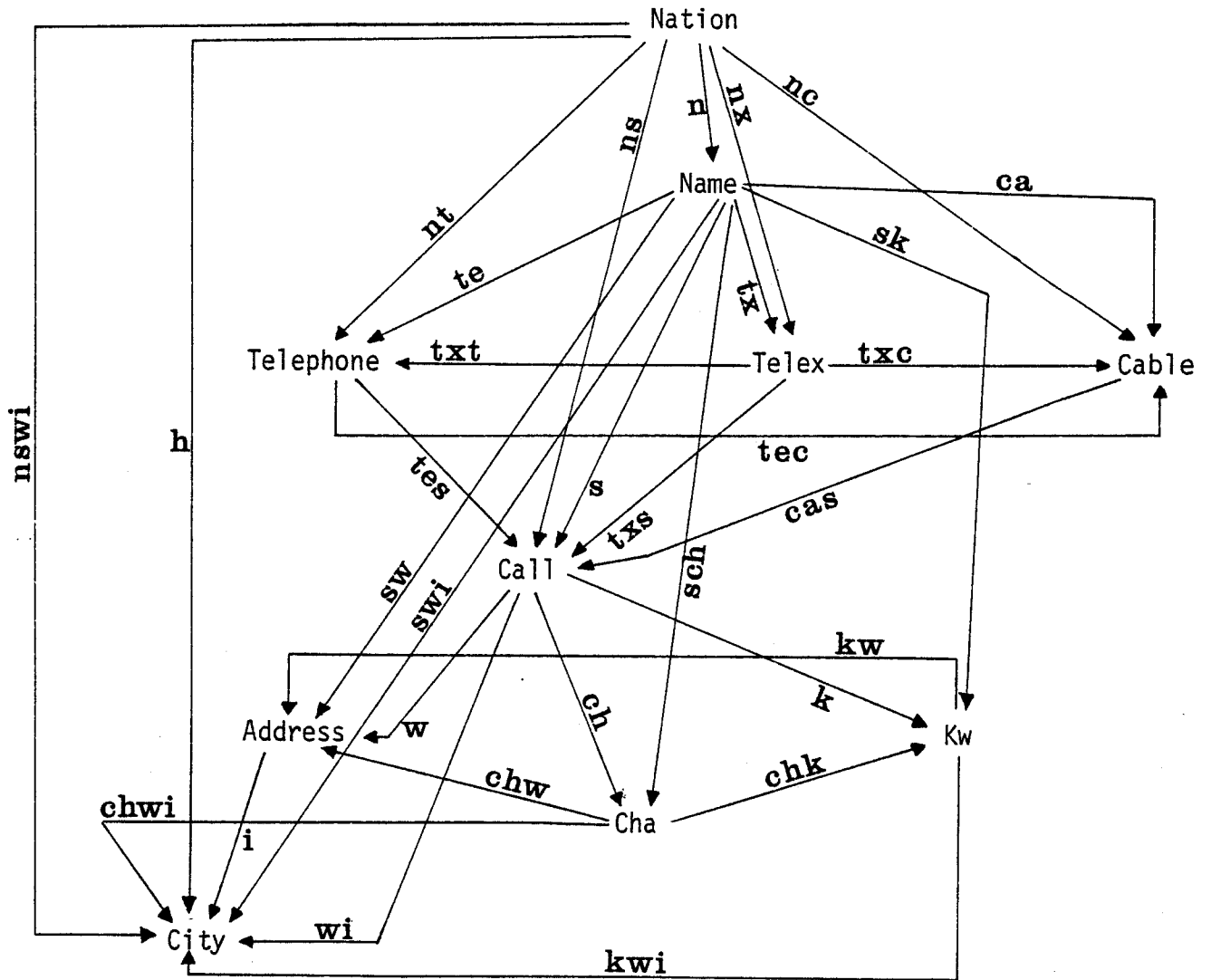


Fig. 6.1 Relational graph H^* describing the conceptual model of the example.

The relational graph H^* corresponding to $I^* = (D, R, \mathcal{P})$ is shown in Fig. 6.1. For such graph we have

$$\psi(I^*) = 1$$

$$\phi(I^*) = 30$$

Let us now apply function $\text{REPRES}(I^*)$ to derive I_0 . The corresponding graph H_0 is shown in Fig. 6.2 and its complexity is the following:

$$\psi(I_0) = 4$$

$$\phi(I_0) = 10$$

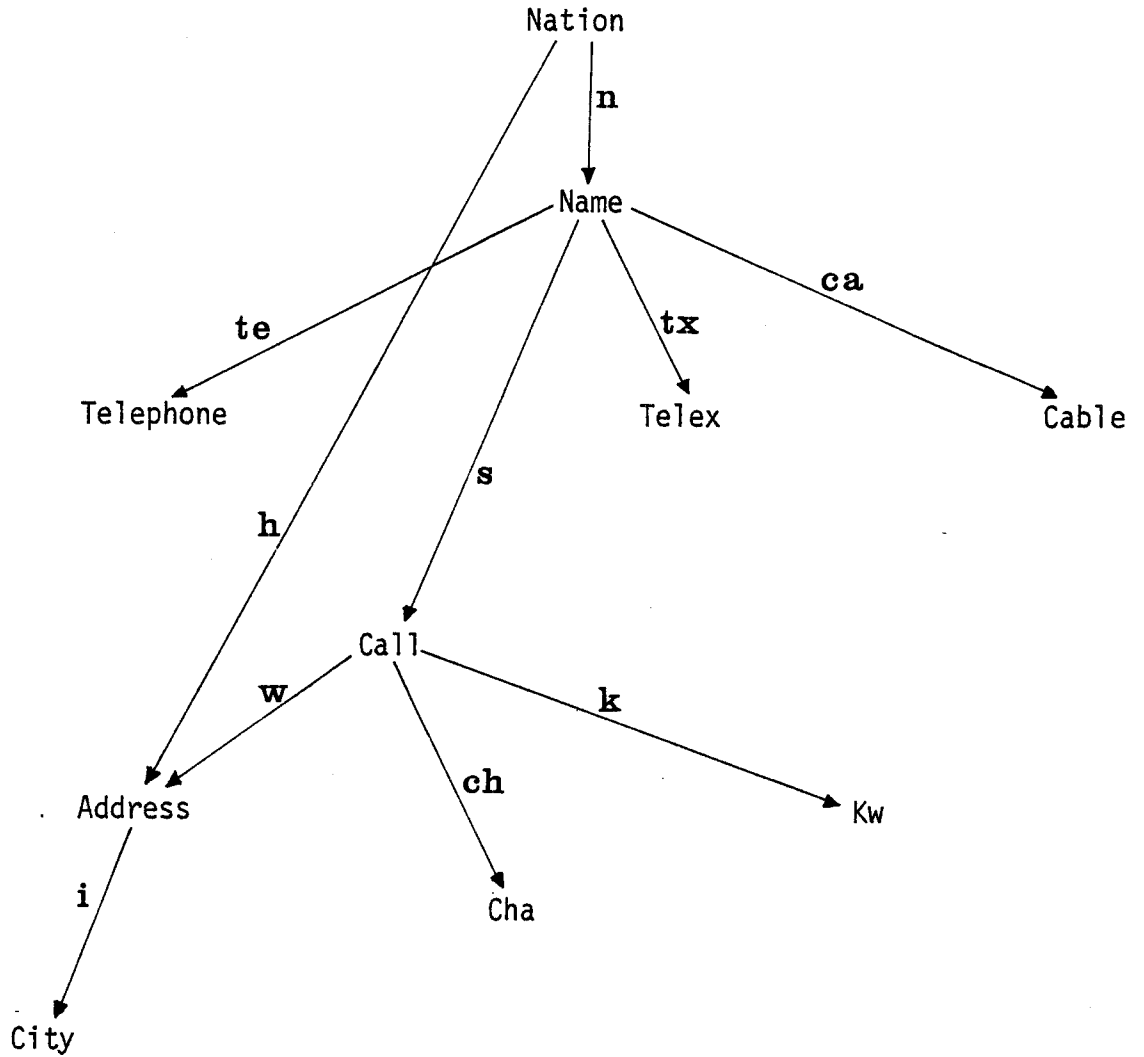


Fig. 6.2 Image of the abstract structure I_0 implementing the conceptual model of the example.

Let us now describe how to construct the redundant implementation SI_ϵ . Consider the skeleton $h(H_0)$ shown in Fig. 6.3(a); this graph has exactly one circuit; therefore we can apply the direct method described in Section 5.3.

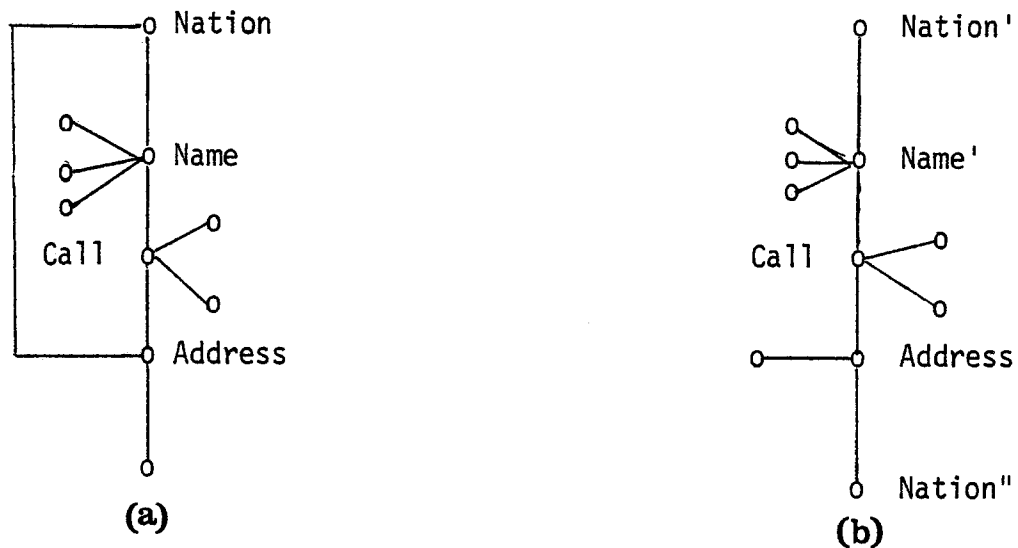


Fig. 6.3 (a) skeleton of the graph H_0 and (b) the corresponding tree obtained by "splitting" the node Nation.

The first step consists of "splitting" a node, e.g. Nation, of two nodes, Nation' and Nation'', obtaining a tree as shown in Fig. 6.3(b). We can now apply the redundant graph approach to the tree; this can be accomplished by finding a node in the tree that generates a best partition, connecting all nodes to it, and then applying the same process to the subtrees rooted in that node. This process is shown in Fig. 6.4 where the dotted lines denote arcs that are added at each step.

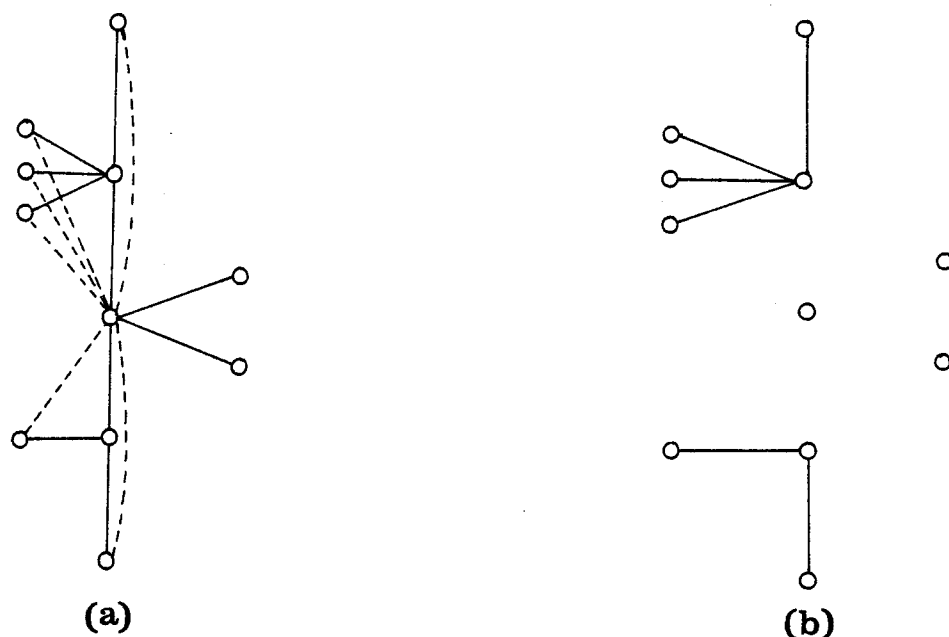


Fig. 6.4 The dotted lines represent the arcs added to the tree of Fig. 6.3(b) when applying the redundant graph approach (a) to the tree, and (b) to the subtrees rooted in the chosen node.

The last step of the redundant graph approach consists of connecting Nation' and Nation'' to all nodes.

Not all the new relations introduced by the above process are needed. In fact we can eliminate all the introduced relations that are not needed for answering a query in at most two steps. For example, the relation corresponding to the edge (Call, Nation'') is not an element of R and is not needed to answer any other query. Therefore $SR_{\epsilon} = R_0 \cup \{ns, wi\}$. (Recall the convention used in this thesis that if a relation belongs to a set so does its inverse.) The corresponding graph SH_{ϵ} is shown in Fig. 6.5.

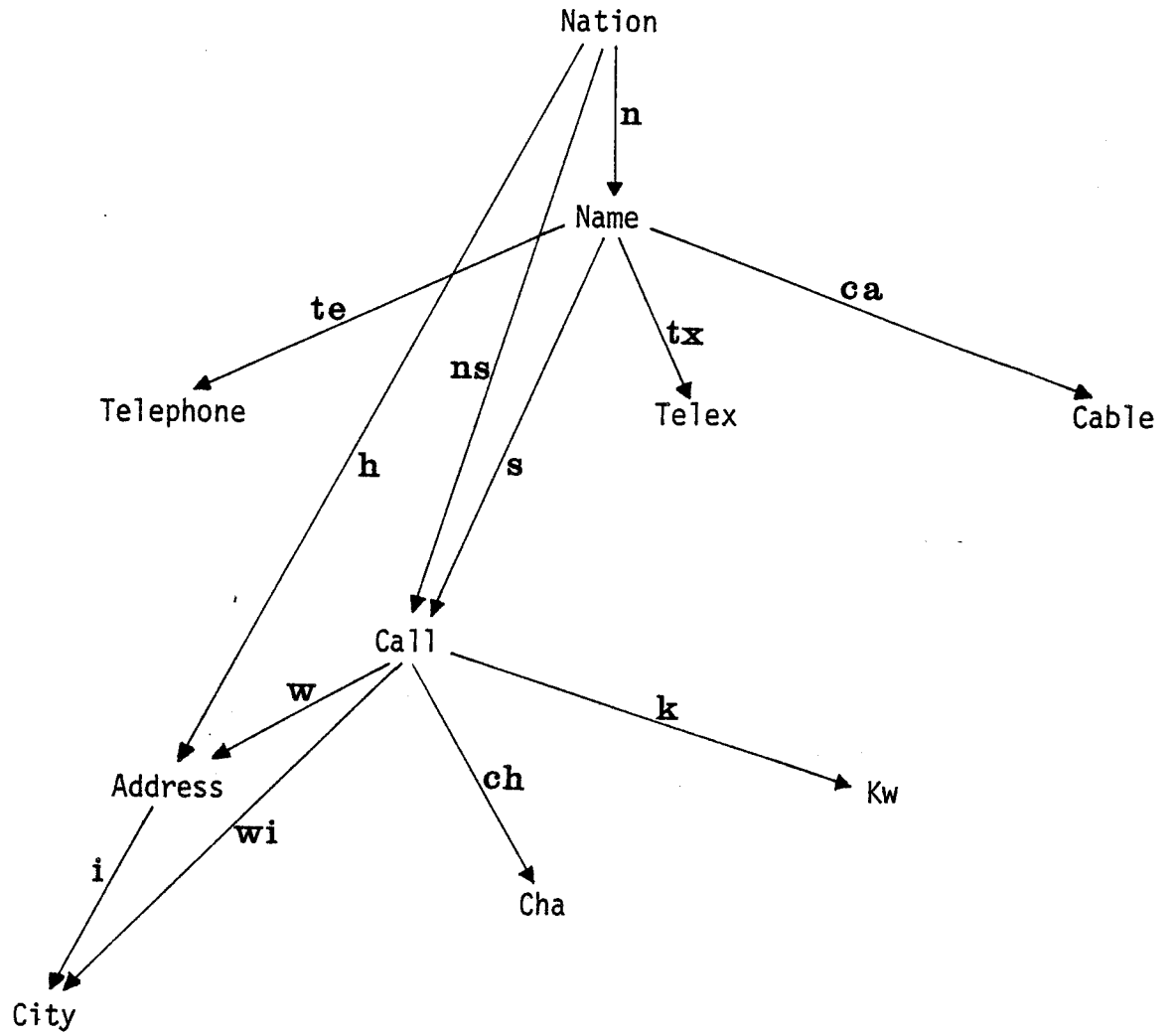


Fig. 6.5 Relational graph SH_{ϵ} obtained by applying the redundant graph approach to the graph H_0 of Fig. 6.2.

For such graph we have

$$\Psi(SI_{\epsilon}) = 2$$

$$\Phi(SI_{\epsilon}) = 11$$

The above results support the theoretical properties analyzed in the previous chapters. In addition, the actual bound on the space complexity of SI_{ϵ} seems to indicate that the redundant graph approach may be more efficient than the theoretical upper bound would imply.

6.3 Other Applications

The results of our research rest mainly in the field of data structure design with obvious applications in data base systems and other large-scale software applications. However several problems, arising in different fields, can either be re-expressed in terms of our research topic or be seen as applications or sub-problems. In this section we will present two problems found in picture processing and communication networks, respectively; and we will show how these problems can be solved using some of the results presented in this thesis.

6.3.1 Constraint Management in Picture Processing

In picture processing, constraints (invariant properties of the class of pictures under consideration) play a very important role, and an explicit and consistent treatment of them can be very advantageous [Mo, Sal]. For example, a systematic specification and utilization of the available constraints could significantly reduce the amount of search in picture recognition. In fact, the specific elements or features searched by recognition routines must be dependent on the elements or features already recognized; for example, the search for the nose, when recognizing human faces, must limit the areas where it may be present; this is further restricted if the position of the mouth and of the eyes have been determined. With a proper use of constraints, the search space can be reduced and only the first elements will be time consuming. Montanari [Mo2] gives a formal treatment of constraints in a

more general way as a network of algebraic equations between possible values assumed by pairs of variables. In addition, a constraint involving many variables is given as an undirected acyclic network of binary relations. Since there are several alternative networks describing the same constraint, there exists the problem of determining an "efficient" network from among them. The efficiency of a network is a trade off between the number of relations in the network and the composition time, where the latter is the number of relations that must be composed to calculate the additional relation among the entities already recognized.

In its original formulation, this problem is still open. Let us use the notation introduced in Chapter 2 to describe the above problem. It is easy to see that the given constraint is exactly the skeleton of the structure I^* , and that any network equivalent to the constraint is a skeleton of a relational graph associated with an abstract implementation of I . Montanari showed that there always exists an acyclic network equivalent to a given constraint; this implies that we are dealing with hierarchical conceptual models only. Let us finally observe that the composition time on a network is exactly the retrieval time on the corresponding relational graph. Therefore, the problem of finding an efficient network of binary relations equivalent to a given constraint is exactly the problem of finding an efficient abstract implementation of a given hierarchical conceptual model. In other words, all the results obtained in Chapters 3 and 4 can be applied to the above problem.

6.3.2 Topological Design of Communication Networks

To design communication networks is a complex task requiring the analysis of different parameters. In particular, time delays, the number of communication links, and reliability are some of the major factors to be considered. A communication network can be represented by a linear graph; this allows us to establish bounds on delay and reliability of the network [Ce]. Time delays can be characterized by measuring the distance between two nodes in the graph; reliability can be characterized by the edge connectivity of the graph. A useful aid to the design of computer communications networks is the study of the so-called "ideal networks", i.e., networks built on the same set of nodes as the "real" one but that minimize certain costs or maximize reliability. In other words, an "ideal" network is a guideline against which the designers may measure their networks. Since ideal networks (also called topological schemas) represent how the topology of the network should look in order to minimize time delays or maximize reliability; then, it is extremely important for such schemas to be as "efficient" as possible. That is, an "efficient" topological schema should optimize as many different variables as possible. Usually the parameters under consideration are the time delay in transmission (distance between two nodes in the graph), reliability (edge-connectivity of the graph), and number of communication lines (number of edges).

The problem of determining an efficient topological schema is a simpler problem than the one analyzed in this thesis. In fact, there are

no semantic constraints on the graph, i.e., all paths between a pair of nodes are equivalent in value. The only difference with the efficiency problem for abstract structures is that we need to take into account also the edge-connectivity of the graph.

In the literature two schemas have been presented; they allow either a strong edge-connectivity and a small distance (at the expenses of the number of communication lines) or a small number of communication lines (at the expenses of the edge-connectivity and node-distance). As a result of this thesis research, a new ideal network, based on a generalization of the graph H^+ , has been proposed [Sa2]. This schema, called the logarithmic star graph, is obtained by constructing $\log n$ star graphs on the same n nodes. The proposed schema has been proved to be more efficient than one of the two others, and to offer a valuable alternative to the other one.

6.4 Conclusions and Extensions

Although several problems have been solved in this thesis, there are some obvious limitations (and thus problems open to research). For example, all the properties of the model of the real world must be expressed as binary relations while, especially in data base applications, many models use n-ary relations. However, from a theoretical point of view, any property involving many data elements simultaneously can be expressed as a set of binary relations [Mol, Ni]; namely, each n-ary relation can be expressed as a relational star graph with $n + 1$ vertices (the extra domain contains an element for each key in the original relation). Therefore, any problem modeled in terms of n-ary relations can be expressed as a problem on binary relation, and thus the results of this thesis can be applied. However, it does not necessarily follow that the solution found in terms of binary relations can easily be transformed into a good solution in terms of n-ary relations. This must still be explored.

The efficiency of abstract structures implementing a conceptual model, has been analyzed with respect to their worst case complexity. However, the time and space complexity observed in some examples in this thesis seem to indicate that the proposed techniques may be more efficient than the theoretical upper bounds would imply. For example, the quadratic worst-case upper bound for the space complexity of the star graph when I^+ is not a minimal perfect implementation, is very far from the values found in practice. To measure the actual effects of the proposed

transformations analytically is a complex task: the values for the variables to be considered (e.g. the topology of the relational graph and the nature of the relations composing the model) depend on each application individually. That is, there can be no closed form solution for space or time without a closed form statement for the inputs. More experimentation for average case behaviour and expected worst case behaviour could probably give a better insight into the complexity of abstract structures. Furthermore experimentation is also valuable for understanding the behaviour for applications which incorporate probability distributions other than the uniform one.

As mentioned in Chapter 5, in practical applications the conceptual model is often either hierarchical (e.g. hierarchical data bases, network of binary constraints equivalent to a n -ary constraint) or very simple in structure (e.g. the conceptual model of Section 6.2). However, all the approaches (worst case, average case and expected worst case as well as consideration of different probability distributions) should also be analyzed for the remaining conceptual models. The work presented here for hierarchical structures has laid a firm foundation towards such research. In particular, the modification of the redundant graph algorithm to graphs with one circuit can first be extended to graphs with several edge-disjoint circuits and then to graphs with arbitrary circuits.

REFERENCES

- Ab... J.R. Abrial, Data Semantics, in Data Base Management Systems, North Holland, Amsterdam, 1974.
- Ash.. W. Ash, E.H. Sibley, TRAMP: An Interpretative Associative Processor with Deductive Capabilities, Proc. 23rd ACM Nat. Conf., 1968.
- Bac.. C.W. Bachman, M. Daya, The Role Concept in Data Model, unpublished manuscript, 1979.
- Bee.. C. Beeri, P.A. Bernstein, N. Goodman, A Sophisticate's Introduction to Database Normalization Theory, Proc. Int. Conf. on Very Large Data Bases, 1978.
- Ben.. J.P. Benson, Structured Programming Techniques, Proc. IEEE Symp. on Computer Software Reliability, 1973.
- Ber.. C. Berge, The Theory of Graphs, Methuen, London, 1962.
- Bil.. H. Biller, On the Equivalence of Data Base Schemas: A Semantic Approach to Data Translation, Inform. Systems 4 (1979).
- Bra.. G. Bracchi, A. Fedeli, P. Paolini, A Relational Data Base Management Information System, Proc. 27th ACM Nat. Conf., 1972.
- Bre.. R.P. Brent, The Parallel Evaluation of General Arithmetic Expressions, JACM 21 (1974).

- Bub.. J.A. Bubenko, S. Berild, E. Lindencrona-Ohlin, S. Nachmens,
From Information requirements to DBTG-Data structures, Proc.
Conf. on Data Abstraction Definition and Management, 1976.
- Ce... V.G. Cerf, D.D. Cowan, R.C. Mullin, R.G. Stanton, Topological Design
Considerations in Computer Communication Networks, Grimsdale and
Kno eds., NATO Advanced Studies, 1974.
- Cha.. L.C. Chang, D.K. Pradhan, A Graph Structural Approach for the
Generalization of Data Management Systems, Information Sciences 12
(1977).
- Cod.. E.F. Codd, A Relational Model of Data for Large Shared Data Banks,
CACM 13 (1970).
- Cre.. A.B. Cremers, T.N. Hibbard, Orthogonality of Information Structures,
Acta Informatica 9 (1978).
- Dal.. A.G. Dale, N.B. Dale, Schema and Occurrence Structure Transformations
in Hierarchical Systems, Proc. SIGMOD Int. Conf. on Management of
Data, 1976.
- Dat.. C.J. Date, An Introduction To Database Systems, 2nd edition,
Addison-Wesley, Reading, 1975.
- deC.. R.L. deCarvalho, A.L. Furtado, A. Pereda-Borquez, A Relational Model
Towards the Synthesis of Data Structures, Technical Report 17/77.
Pontificia Universidade Catolica do Rio de Janeiro, 1977.

- Deh.. C. Deheneffe, H. Hennembert, NUL: A Navigational User's Language For a Network Structured Database, Proc. Int. Conf. on Management of Data, 1976.
- Dij.. E.W. Dijkstra, Notes on Structured Programming, in Structured Programming, Academic Press, New York, 1972.
- DIm.. M.E. D'Imperio, Data Structures and Their Representation in Storage, in Annual Rev. in Auto. Prog. 5, Pergamon Press, Oxford, 1969.
- Fe... J.A. Feldman, P.D. Rovner, An Algol-Based Associative Language, CACM 12 (1969).
- Fr... J.M. Frost (ed.), World Radio TV Handbook 33, Billbord, London, 1979.
- Gh... S.P. Ghosh, M.E. Senko, String Path Search Procedures for Data Base Systems, IBM J. of Res. and Develop. 18 (1974).
- Go1.. M.K. Goldberg, The Diameter of a Strongly Connected Graph, (in Russian), Dokl. Akad. Nauk. SSSR 170 (1966).
- Got.. C.C. Gotlieb, A.L. Furtado, Data Schemata Based on Directed Graphs, Int. J. Comput. Inf. Sci. 8 (1979).
- Gr... G. Gratzer, Universal Algebra, Van Nostrand, Princeton, 1968.
- Gut.. J. Guttag, Abstract Data Types and the Development of Data Structures, CACM 20 (1977).

- Ham.. M. Hammer, Data Abstractions for Data Bases, Proc. Conference on Data Abstraction Definition and Management, 1976, 58-59.
- Har.. W.T. Hardgrave, Theoretical Aspects of Boolean Operations on Tree Structures and Implications for Generalized Data Management, Computational Centre Report TSN-26, University of Texas at Austin, 1972.
- Ho.. C.A.R. Hoare. Notes on Data Structuring, in Structured Programming, Academic Press, New York, 1972.
- Iso.. Open System Interconnections, ISO/TC97/SC16, Document 117, 1978.
- Ka... B.K. Kahn, A Method for Describing Information Required by the Data Base Design Process, Proc. SIGMOD Int. Conf. on Management of Data, 1976.
- La... B. Langefors, Theoretical Aspects of Information Systems for Management, Proc. IFIP, 1974.
- Le... R.E. Levein, M.E. Maron, A Computer System for Inference Execution and Data Retrieval, CACM 10 (1967).
- Lis.. B.H. Liskov, S.N. Zilles, Programming with Abstract Data Types, SIGPLAN Notices 9 (1974).
- Low.. J.R. Low, Automatic Data Structure Selection: An Example and Overview, CACM 21 (1978).

- Maj.. M.E. Majester, Extended Directed Graphs: A Formalism for Structured Data and Data Structures, Acta Informatica 8 (1977).
- McC.. W.A. McCuskey, On Automatic Design of Data Organization, Proc. AFIPS 37, 1970.
- McGe. W.C. McGee, A Contribution to the Study of Data Equivalence, in Data Base Management Systems, North Holland, Amsterdam, 1974.
- McGo. C.L. McGowan, J. Kelly, Top-down Structured Programming Techniques, Petrocelli-Charter, New York, 1975.
- Me... G. Mealy, Another Look at Data, Proc. AFIPS 31, 1967.
- Mi... M.F. Mitoma, K.B. Irani, Automatic Data Base Schema Design and Optimization, Proc. Int. Conf. on Very Large Data Bases, 1975.
- Mo1.. U. Montanari, Network of Constraints: Fundamental Properties and Applications to Picture Processing, Information Sciences 7 (1974).
- Mo2.. U. Montanari, Data Structures, Program Structures and Graph Grammars, Informatica 7 (1977).
- Mun.. R. Munz, The WELL System: A Multi-user Database System Based on Binary Relationships and Graph-Pattern-Matching, Information Systems 3 (1978).
- Ne... E.J. Neuhold, Formal Properties of Data Bases, in Foundations of Computer Science, DeBakker ed., Mathematisch Centrum, Amsterdam, 1975.

- Ray.. F.B. Ray, Directed Graph Structures for Data Base Management: Theory, Storage and Algorithms, Computational Center Report TSN-31, University of Texas at Austin, 1972.
- Ros.. A.L. Rosenberg, Data Graphs and Addressing Schemes, J. Comp. System Sci. 5 (1971), 193-238.
- Sal.. N. Santoro, Economic Handling of Constraints in Picture Processing, Proc. IEEE Conf. on Picture Data Description and Management, 1977.
- Sa2.. N. Santoro, On the Topological Design of Computer Communication Networks, Proc. 6th Int. Symp. on Computers Electronics and Control, 1978.
- Sch.. L.S. Schneider, A Relational View of the Data Independent Accessing Model, Proc. SIGMOD Int. Conf. on Management of Data, 1976.
- Se1.. M.E. Senko, E.B. Altman, M.M. Astrahan, P.L. Fehder, Data Structures and Accessing in Data Base Systems, IBM Syst. J. 12 (1973).
- Se2.. M.E. Senko, DIAM II: The Binary Infological Level and its Data Base Language -FORAL, Proc. Conf. on Data Abstraction Definition and Management, 1976.
- Su... B. Sundgren, An Infological Approach to Data Bases, PhD Thesis, University of Stockholm, 1973.
- To1.. F.W. Tompa, Choosing an Efficient Internal Schema, in Systems for Large Data Bases, North Holland, New York, 1976.

- To2.. F.W. Tompa, Data Structure Design, in Data Structures Computer Graphs and Pattern Recognition, Academic Press, New York, 1978.
- Ts1. D.C. Tschritzis, A. Klug (eds.), The ANSI/X3/SPARC DBMS Framework, Information Systems 3 (1978).
- Ts2. D.C. Tschritzis, F.H. Lochovsky, Hierarchical Data Base Management, Computing Surveys 8 (1976).
- Tsu.. T. Tsuji, J. Toyoda, K. Tanaka, Relational Data Graphs and Some Properties of Them, J. Comp. System Sci. 15 (1977).
- Wir.. N. Wirth, Program Development by Stepwise Refinements, CACM 14 (1971).

APPENDIX A

EXTENDED ALGEBRA OF BINARY RELATIONS

Relations

Given two arbitrary non empty sets D_1, D_2 , a binary relation r from D_1 to D_2 is a set $r \subseteq (D_1 \cup \delta) \times (D_2 \cup \delta)$ of ordered pairs such that

- (i) $\forall x \in D_1 \{(\exists y \in D_2 \text{ such that } (x,y) \in r) \text{ or } ((x,\delta) \in r)\}$
- (ii) $\forall y \in D_2 \{(\exists x \in D_1 \text{ such that } (x,y) \in r) \text{ or } ((\delta,y) \in r)\}$
- (iii) $(\delta,\delta) \in r$

where $\delta \notin D_1 \cup D_2$ is a distinguished element of the algebra and is called null element.

The sets D_1 and D_2 are called the left and the right domain or r , respectively, and denoted by $ld[r] = D_1$ and $rd[r] = D_2$, or by $[r] = (D_1, D_2)$.

Inverse

Given a relation r , $[r] = (D_1, D_2)$, the inverse of r , denoted by r^{-1} , is the set $r^{-1} = \{(y,x) | (x,y) \in r\}$. Of course, $[r^{-1}] = (D_2, D_1)$ and $(r^{-1})^{-1} = r$.

Composition

Given a set of data elements D , let R_D denote the set of all possible binary relations between subsets of D . The basic operation on R_D is the composition of relations:

given $r_1, r_2 \in R$, where $[r_1] = (D_{11}, D_{12})$ and $[r_2] = (D_{21}, D_{22})$, the composition of r_1 with r_2 , denoted by $r_1 \cdot r_2$ is recursively defined as follows:

(i) $\forall x \in D_{11}, \forall y \in D_{22}$, if $\exists z \in D_{12} \cap D_{21}$ such that $(x, z) \in r_1$ and $(z, y) \in r_2$, then $(x, y) \in r_1 \cdot r_2$;

(ii) $\forall x \in D_{11}$ if $\forall y \in D_{22} (x, y) \notin r_1 \cdot r_2$ then $(x, \delta) \in r_1 \cdot r_2$;

(iii) $\forall y \in D_{22}$ if $\forall x \in D_{11} (x, y) \notin r_1 \cdot r_2$ then $(\delta, y) \in r_1 \cdot r_2$.

The above operation is such that $r = r_1 \cdot r_2$ is a relation and $[r] = (D_{11}, D_{22})$.

By $\lambda = \{(\delta, \delta)\}$ we shall denote the empty relation, and by $i = \{(x, x) | x \in D\}$ the identity relation.

Given $r_1, r_2, r_3 \in R$, the following properties hold:

$$(1) \quad r_1 \cdot (r_2 \cdot r_3) = (r_1 \cdot r_2) \cdot r_3 ;$$

$$(2) \quad r_1 \cdot i = i \cdot r_1 = r_1 ;$$

$$(3) \quad r_1 \cdot \lambda = \lambda \cdot r_1 = \lambda ;$$

$$(4) \quad (r_1 \cdot r_2)^{-1} = r_2^{-1} \cdot r_1^{-1} .$$

Sequences

Given a set of relations $R_i \subseteq R_D$, we will denote by $S(R_i)$ the set of all the finite sequences of elements of R_i and their inverses such that:

- (1) in each sequence there is at most one occurrence of any relation;
- (2) in each sequence $q \in S(R_i)$, if $r \in q$ ($r^{-1} \in q$) then $r^{-1} \notin q$ ($r \notin q$).

Given $p, q \in S(R_i)$, $p = \langle p_1 p_2 \dots p_n \rangle$ and $q = \langle q_1 q_2 \dots q_m \rangle$, we will say that p and q are equivalent, $p \equiv q$, if noncoincidentally $p_1 \cdot p_2 \cdot \dots \cdot p_n = q_1 \cdot q_2 \cdot \dots \cdot q_m$.

We extend the definition of right and left domain to sequences as follows: given $p = \langle p_1 p_2 \dots p_n \rangle \in S(R_i)$, then $rd[p] = rd[p_n]$ and $ld[p] = ld[p_1]$.

Closure

Given $R_i \subseteq R_D$, the transitive closure of R_i , denoted by $(R_i)^*$, is the set of all distinct relations that can be obtained through composition of elements of R_i .

Domain

Given a set of relations $R_i \subseteq R_D$, we will denote by $C(R_i)$ the set of domains $C(R_i) = \{D_j \subseteq D \mid r \in R_i (D_j = rd[r] \text{ or } D_j = ld[r])\}$.

APPENDIX B

PROPERTIES OF RELATIONAL GRAPHS

A relational graph $H = (V, L, E)$ is a directed edge-labelled multigraph, where V is a non-empty set of domains and L is a non-empty set of labels naming binary relations; given $x, y \in V$, $z \in L$, there is an edge e from x to y labeled z , $e = (x, y; z)$, if and only if $ld[z] = x$ and $rd[z] = y$. If $(x, y; z)$ is an edge in E , then we say that $(y, x; z^{-1})$ is an inverse edge in E . Given a finite sequence of edges and inverse edges, we will say that there is a reversed occurrence of an edge $(x, y; z)$ if $(y, x; z^{-1})$ is in the sequence.

Given a finite sequence of edges or inverse edges of a relational graph H , such that there is at most one occurrence (reversed or not) of each edge in the sequence, $p = \langle (d_1, d_2; r_1), (d_2, d_3; r_2), \dots, (d_m, d_{m+1}; r_m) \rangle$, we say that the sequence $\langle r_1 r_2 \dots r_m \rangle$ is a relational path (or simply path) on the relational graph; d_1 is called origin of the path, and d_m is called end of the path. A cycle is a relational path $\langle r_1 \dots r_m \rangle$ where $d_1 = d_{m+1}$. Obviously, if $\langle r_1 \dots r_m \rangle$ is a cycle, then also $\langle r_i r_{i+1} \dots r_m r_1 \dots r_{i-1} \rangle$ is a cycle, $1 \leq i \leq m$.

Given a relational graph H , the skeleton $h(H)$ of H is the undirected multigraph (or simply graph) obtained from H eliminating the orientation of the edges, ignoring the labels and considering each edge together with its inverse edge as one simple arc. Formally, a graph

G is a couple $G = (N,A)$ where N is a non-empty set of nodes and $A \subseteq N \times N$ is a set of arcs.

The diameter of a graph G is the maximum distance between two nodes in N , and is denoted by $d(G)$. The degree of a node x is the number of arcs incident on x , and is denoted by $g[x]$.

Given a finite sequence of arcs, $q = \langle (n_1, n_2), (n_2, n_3), \dots, (n_j, n_{j+1}) \rangle$, $n_i \in N$, $1 \leq i \leq j + 1$, we will say that p is a circuit if $n_{j+1} = n_1$.

APPENDIX C

DATA FOR THE FINAL EXAMPLE

n = {(Canada, CBC), (Canada, CTV), (Canada, TVA),
(Canada, ORTQ), (Canada, OECA)}.

te = {(CBC, 613-731-3111), (CTV, 414-928-6000), (OECA, 416-484-4600)}.

tx = {(CBC, 053-4260), (CTV, 06-22080)}.

ca = {(CBC, broadcast)}.

s = {(CBC, CFLA), (CBC, CBYT), (CBC, CBLNT), (CBC, CBNT),
(CBC, CBCT), (CBC, CBHT), (CBC, CBIT), (CBC, CBMT),
(CBC, CBOT), (CBC, CBLT), (CBC, CBET), (CBC, CBWT),
(CBC, CBKRT), (CBC, CBKST), (CBC, CBRT), (CBC, CBXT),
(CBC, CBUT), (CBC, CBAFT), (CBC, CBGAT), (CBC, CBFT),
(CBC, CBVT), (CBC, CBRT-TV), (CBC, CBOFT), (CBC, CBLFT),
(CBC, CBWFT), (CBC, CBXFT), (CBC, CBUFT), (CTV, CJON),
(CTV, CJCH), (CTV, CJCB), (CTV, CKCW), (CTV, CFCF),
(CTV, CKCO), (CTV, CJOH), (CTV, CKSO), (CTV, CFTO),
(CTV, CKY), (CTV, CKCK), (CTV, CFQC), (CTV, CFCN),
(CTV, CFRN), (CTV, BCTV), (CTV, CHEK), (CTV, CHFD),
(CTV, CICC), (CTV, CITL), (CTV, CKCY), (TVA, CFCM),
(TVA, CJPM), (TVA, CHLT), (TVA, CHEM), (TVA, CIMT),

(TVA, CFTM), (TVA, CFER), (TVA, CHOT), (ORTQ, CIVM),
(ORTQ, CIVQ), (ORTQ, CIVO), (OECA, CICO)}.

w = {(CFLA, P.O. Box 925-Stn. "A"), (CBYT, P.O. Box 610),
(CBNLT, P.O. Box 576), (CBNT, P.O. Box 12010-Stn "A"),
(CBCT, P.O. Box 2230), (CBHT, P.O. Box 3000),
(CBIT, P.O. Box 700), (CBMT, P.O. Box 6000),
(CBOT, P.O. Box 3220), (CBLT, P.O. Box 500),
(CBET, P.O. Box 1609), (CBWT, P.O. Box 160),
(CBKRT, 1840 McIntyre_St),
(CBKST, 5th Floor/CN_Tower/1st_Ave_South),
(CBRT, P.O. Box 2640), (CBXT, P.O. Box 555),
(CBUT, P.O. Box 4600), (CBAFT, P.O. Box 950),
(CBGAT, P.O. Box 2000), (CBFT, P.O. Box 6000),
(CBVT, P.O. Box 10400), (CBOFT, P.O. Box 3220),
(CBRT-TV, 273 ST_Jean_Baptiste_Ouest),
(CBLFT, P.O. Box 500-Stn "A"), (CBWFT, P.O. Box 160),
(CBXFT, P.O. Box 555), (CBUFT, P.O. Box 4600),
(CJON, P.O. Box 2020), (CJCH, 2885 Robie Str),
(CJBC, P.O. Box 469), (CKCW, P.O. Box 5004),
(CFCF, 405 Ogilvy Ave), (CKCO, 864 King St West),
(CJOH, 1500 Merivale Rd), (CKSO, P.O. Box 400),
(CFTO, P.O. Box 9), (CKCK, P.O. Box 2000),
(CFOC, 216 First Avenue North), (CKY, Polo Park),
(CFCN Broadcast House), (BCTV, P.O. Box 4700),

(CFRN, 18520_Stony_Plain_Rd), (CHEK, 3693_Epsom_Dr),
(CHFD, 87 North Hill St), (CICC, 95 East Broadway),
(CITL, 5026-50th_Street), (CKCY, P.O. Box 370),
(CFCM, C.P. 2026), (CJPM, C.P. 600),
(CHLT, 3330_rue_King_Ouest), (CFER, C.P. 590),
(CHEM, 1400_rue_Des_Cypres), (CIMT, 1_rue_Frontenac),
(CFTM, C.P. 170), (CHOT, C.P. 4010)}.

ch = {(CFLA, 8), (CBYT, 5), (CBNLT, 13), (CBNT, 8), (CICO, 19),
(CBCT, 13), (CBHT, 3), (CBIT, 5), (CBMT, 6), (CBOT, 4),
(CBLT, 5), (CBET, 9), (CBWT, 6), (CBKRT, 4), (CBKST, 11),
(CBRT, 9), (CBXT, 5), (CBUT, 2), (CBGAT, 9), (CBAFT, 11),
(CBFT, 2), (CVVT, 11), (CBRT, 3), (CBOFT, 9), (CBFLT, 25),
(CBWFT, 3), (CBXFT, 11), (CBUFT, 26), (CJON, 6), (CVO, 3),
(CJCH, 5), (CJBC, 4), (CKCW, 2), (CFCF, 12), (CKCO, 13),
(CJOH, 13), (CKSO, 5), (CFTO, 9), (CKY, 7), (CKCK, 2),
(CFQC, 8), (CFCN, 4), (CFRN, 3), (BCTV, 8), (CHEK, 6),
(CHFD, 4), (CICC, 10), (CITL, 4), (CKCY, 2), (CFCM, 4),
(CJPM, 6), (CHLT, 7), (CHEM, 8), (CIMT, 9), (CFTM, 10),
(CFER, 11), (CHOT, 40), (CIVM, 17), (CIVQ, 15)}.

k = {(CFLA, .83), (CBYT, 10.6), (CBNLT, .214), (CBNT, 196),
(CBCT, 178), (CBHT, 56), (CBIT, 54), (CBMT, 100),
(CBOT, 100), (CBLT, 84), (CBET, 178), (CBWT, 100),
(CBKRT, 140), (CBKST, 325), (CBRT, 178), (CBXT, 318),

(CBUT, 47.6), (CBAFT, 163), (CBGAT, 153), (CBFT, 100),
(CBVT, 173), (CBRT, 100), (CBOFT, 128), (CBFLT, 234),
(CBWFT, 59), (CBXFT, 90), (CBUFT, 105), (CJON, 110),
(CJCH, 100), (CJBC, 180), (CKCW, 25), (CFCF, 325),
(CKCO, 325), (CJOH, 178), (CKSO, 100), (CFTO, 325),
(CKY, 325), (CKCK, 100), (CFQC, 325), (CFCN, 100),
(CFRN, 180.3), (BCTV, 164), (CHEK, 60), (CHFD, 56),
(CICC, 56), (CITL, 82), (CKCY, 100), (CFCM, 100),
(CJPM, 100), (CHLT, 316), (CHEM, 125), (CIMT, 49),
(CFTM, 325), (CFER, 325), (CIVM, 242), (CIVQ, 259),
(CIVO, 530), (CICO, 108)}.

i = {(P.O. Box 925-Stn. "A", Goose Bay-Lab),
(P.O. Box 610, Corner Brook-Nfld.), (P.O. Box 576, Labrador
City-Nfld.), (P.O. Box 1210-Stn. "A", St. Johns-Nfld.),
(P.O. Box 2230, Charlottetown-P.E.I.),
(P.O. Box 3000, Halifax-N.S.), (P.O. Box 700, Sydney-N.S.),
(P.O. Box 6000, Montreal-Que.), (P.O. Box 3220, Ottawa, Ont.),
(P.O. Box 500, Toronto-Ont.), (P.O. Box 1609, Windsor-Ont.),
(P.O. Box 160, Winnipeg-Man.), (1840_McIntyre_St., Regina-Sask.),
(5th_Floor/CN Tower/1st_Ave_South, Saskatoon-Sask.),
(P.O. Box 2640, Calgary-Alta.),
(P.O. Box 555, Edmonton-Alta.),
(P.O. Box 4600, Vancouver, B.C.), (P.O. Box 950, Moncton-N.B.),
(P.O. Box 2000, Matane-Que.),

(P.O. Box 6000, Montreal-Que.), (P.O. Box 10400, St. Foy-Que.),
(273_ST_Jean_Baptiste_Ouest, Rimouski-Que.),
(P.O. Box 3220, Ottawa-Ont), (P.O. Box 500-Stn "A", Toronto-Ont.),
(P.O. Box 160, Winnipeg-Man.),
(P.O. Box 555, Edmonton-Alta), (P.O. Box 4600, Vancouver-B.C.),
(P.O. Box 2020, St. John's-Nfld.),
(2885_Robie_Str., Halifax-N.S.), (P.O. Box 469, Sydney-N.S.),
(P.O. Box 5004, Moncton-N.B.),
(405_Ogilvy_Ave., Montreal-Que.),
(864_King_St._West, Kitchener-Ont.),
(1500_Merivale_Rd., Ottawa-Ont.), (P.O. Box 400, Sudbury-Ont.),
(P.O. Box 9, Toronto-Ont.), (Polo_Park, Winnipeg-Man.),
(P.O. Box 2000, Regina-Sask.),
(216_First_Avenue_North, Saskatoon-Sask.),
(Broadcast_House, Calgary-Alta.),
(18520_Stony_Plain_Rd., Edmonton-Alta.),
(P.O. Box 4700, Vancouver-B.C.),
(3693_Epsom_Dr., Saanich-B.C.),
(87_North_Hill_St., Thunder Bay-Ont.),
(95_East_Broadway, Yorkton-Sask.),
(5026-50th Street, Lloydminster-Sask.-Alta.),
(P.O. Box 370, Sault_St. Marie-Ont.), (C.P. 2026, Quebec-Que.),
(C.P. 600, Chicoutimi-Que.),
(3330_rue_King_Ouest, Sherbrooke-Que.),
(1400_rue_Des_Cypres, Trois_Rivieres-Que.),
(1_rue_Frontenac, Riviere_du_Loupe-Que.),
(C.P. 170, Montreal-Que.), (C.P. 590, Rimouski-Que.),

(C.P. 4010, Hull-Que.)}.

h = {(CBC, P.O. Box 8478), (CTV, 42_Charles_Str. East),
(TVA, 1600_de-Maisonneuve), (ORTQ, 100_rue_Fullum),
(OECA, P.O. Box 200_Station "A")}.

wi = {(CFLA, Goose Bay-Lab.), (CBYT, Corner Brook-Nfld.),
(CBNLT, Labrador City-Nfld.), (CBNT, St. Johns-Nfld.),
(CBCT, Charlottetown-P.E.I.), (CBHT, Halifax-N.S.),
(CBIT, Sydney-N.S.), (CBMT, Montreal-Que.),
(CBOT, Ottawa-Ont.), (CBLT, Toronto-Ont.),
(CBET, Windsor-Ont.), (CBWT, Winnipeg-Man.),
(CBKRT, Regina-Sask.), (CBKST, Saskatoon-Sask.),
(CBRT, Calgary-Alta.), (CBXT, Edmonton-Alta.),
(CBUT, Vancouver-B.C.), (CBAFT, Moncton-N.B.),
(CBGAT, Matane-Que.), (CBFT, Montreal-Que.),
(CBVT, St. Foy-Que.), (CBRT, Rimouski-Que.),
CBOFT, Ottawa-Ont.), (CBFLT, Toronto, Ont.),
(CBWFT, Winnipeg-Man.), (CBXFT, Edmonton-Alta.),
(CBUFT, Vancouver-B.C.), (CJON, St. John's-Nfld.),
(CJCH, Halifax-N.S.), (CJBC, Sydney-N.S.),
(CKCW, Moncton-N.B.), (CFCF, Montreal-Que.),
(CKCO, Kitchener-Ont.), (CJOH, Ottawa-Ont.),
(CKSO, Sudbury-Ont.), (CFTO, Toronto-Ont.),
(CKY, Winnipeg, Man.), (CKCK, Regina-Sask.),

(CFQC, Saskatoon-Sask.), (CFCN, Calgary-Alta.),
(CFRN, Edmonton-Alta.), (BCTV, Vancouver-B.C.),
(CHEK, Saanich-B.C.), (CHFD, Thunder Bay-Ont.),
(CICC, Yorkton-Sask.), (CITL, Lloydminster-Sask.),
(CKCY, Sault St. Marie-Ont.), (CFCM, Quebec-P.Q.),
(CJPM, Chicoutimi-P.Q.), (CHLT, Sherbrooke-P.Q.),
(CHEM, Trois_Rivieres-P.Q.), (CIMT, Riviere_du_Loup_P.Q.),
(CFTM, Montreal-P.Q.), (CFER, Rimouski-P.Q.),
(CHOT, Hull-PQ.), (CIVM, Montreal-P.Q.),
(CIVO, Hull-P.Q.), (CICO, Toronto-Ont.),
(CIVQ, Quebec-P.Q.)}.

ns = {(Canada, CFLA), (Canada, CBYT), (Canada, CBNLT),
(Canada, CBNT), (Canada, CBCT), (Canada, CBHT),
(Canada, CBIT), (Canada, CBMT), (Canada, CBOT),
(Canada, CBLT), (Canada, CBET), (Canada, CBWT),
(Canada, CBKRT), (Canada, CBKST), (Canada, CBRT),
(Canada, CBXT), (Canada, CBUT), (Canada, CBAFT),
(Canada, CBGAT), (Canada, CBFT), (Canada, CBVT),
(Canada, CBRT), (Canada, CBOFT), (Canada, CBFLT),
(Canada, CBWFT), (Canada, CBXFT), (Canada, CBUFT),
(Canada, CJON), (Canada, CJCH), (Canada, CJBC),
(Canada, CKCW), (Canada, CFCF), (Canada, CKCO),
(Canada, CJOH), (Canada, CKSO), (Canada, CTFO),
(Canada, CKY), (Canada, CKCK), (Canada, CFQC),

(Canada, CFCN), (Canada, CFRN), (Canada, BCTV),
(Canada, CHEK), (Canada, CHFD), (Canada, CICC),
(Canada, CITL), (Canada, CKCY), (Canada, CFCM),
(Canada, CJPM), (Canada, CHLT), (Canada, CHEM),
(Canada, CIMT), (Canada, CFTM), (Canada, CFER),
(Canada, CHOT)}.