TOWARDS THE EFFECTIVE IMPLEMENTATION OF DESCRIPTIVE STORAGE

by

James W. Welch

Research Report CS-75-06

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

February, 1975

TOWARDS THE EFFECTIVE IMPLEMENTATION OF DESCRIPTIVE STORAGE

BY

JAMES W. WELCH

A THESIS

PRESENTED TO THE

UNIVERSITY OF WATERLOO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF APPLIED ANALYSIS AND COMPUTER SCIENCE

FACULTY OF MATHEMATICS

MARCH 1974

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

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

Signature James W. Welch

ACKNOWLEDGEMENTS

I would like to thank my research advisor, Professor J. Wesley Graham, for his interest in this research. Professor Graham was responsible for my introduction to computer science and has supervised my development in this field. Throughout the research, his criticism and advice was invaluable. In summary, it is doubtful that this thesis would have been accomplished, were it not for his support and encouragement.

I would also like to thank my colleagues, L. S. Law, R. D. Truman, and M. S. Doyle, for their criticisms and discussions with regard to many ideas related to this thesis.

Abstract

The information in many file systems may be divided into two classes: descriptive storage and basic storage. Descriptive storage is that information whose only inherent value is to facilitate access to the basic information. This thesis is concerned with the implementation of Set-theoretical Descriptive Storage (STDS). Informally, an STDS is a file in which the descriptive storage is composed of sets. It is shown that several well-known file structures are examples of STDS files; i.e., inverted and multilist files. Retrieval in an STDS is analyzed in a general manner. A theory is developed to formally specify how to use the sets in descriptive storage for retrieval purposes. The problem of choosing the best collection of sets for a particular STDS is analyzed. It is proposed that the branch-and-bound method be adapted to select the optimal collections. Several particular STDS systems are considered. It is shown that retrieval in these systems can be generalized in order to process more retrieval requests and in some cases can be made more efficient. How to optimize some of these systems is also indicated. Previously, no optimization methods existed for these systems.

PREFACE

In each of the first four chapters of this thesis a summary of the contents of that appropriate chapter is included in the introductory section. The purpose of these sections is to indicate those areas which are background information and those in which new results are obtained. It is hoped that this approach will facilitate the reading of the thesis. A brief outline of the thesis is as follows.

CHAPTER 1

This chapter introduces the concepts upon which our research is based. It provides a review of the literature and introduces the concept of descriptive storage. Descriptive storage is that information in a file system which is used to facilitate access to the primary information in the file. An idealized model of cost is developed for systems which use descriptive storage. A general class of descriptive-storage systems is defined, called Set-theoretical Descriptive S torage (STDS). STDS systems include several well-known file systems such as inverted files and multilist files.

CHAPTER 2

This chapter is an analysis of the STDS environment with particular emphasis upon retrieval. For an arbitrary Boolean expression of attributes, it is demonstrated how to determine the records for which such a query is true. The costs of operating an STDS system are analyzed and a model of operation is proposed.

CHAPTER 3

This chapter is an analysis of how to select the sets which compose descriptive storage in an STDS system. It is proposed that the well-known Branch-and-Bound method be applied to select an optimal collection of sets, given a collection of candidates. Several heuristic approaches are recommended for the implementation of this algorithm. The problem of choosing the candidate collection is analyzed.

CHAPTER 4

The results in chapters two and three are applied to the following file systems: inverted files, multilist files, and a structure proposed by Wong and Chiang. Several results unique to particular systems are also presented. An STDS system for future research is proposed.

CHAPTER 5

We conclude our remarks in this chapter.

Table of Contents

Chapter 1: Fundamental concepts

1.1 Introduction

This thesis is concerned with the implementation of file systems stored on direct-access storage devices. A review of the literature in this area is presented in section 1.2. Briefly, the historical development is as follows. The first file systems were implemented using magnetic tape or punched cards. The cost of operating these systems is largely proportional to the number of times the files are sequentially read. In order to reduce the cost of these systems, the various processing requests are collected into batches. The requests in each batch are satisfied by reading the file once and performing the indicated operations concurrently. In many cases it is inconvenient for the users of such a system to have to wait for the period of time that is required to collect a batch of requests. The development of direct-access storage devices, principally disks and drums, permitted files to be processed in a non-sequential manner. Records in a file can be accessed randomly using these devices. A number of file systems were developed to take advantage of this feature. These file systems include inverted files, multilist files, and indexed files.

In addition to the data records, many systems also maintain a supplementary collection of information which describes the content of the data records. The accession

algorithms can use this description in order to determine which records are to be accessed in order to satisfy a given processing request. Consider, for example, a student-record file. A list of pointers to records of students living in residence could be maintained. This list could be utilized while producing a report of all students who had not paid their residence fees.

It is apparent that two kinds of information is maintained by these systems. First, there is the information which is of primary concern to the user(s), called basic storage. In the preceding example, the basic storage is the file of data records. Secondly, there is descriptive storage: that information whose only inherent value is to facilitate access to the basic information. In the preceding example, the list of pointers is an illustration of descriptive storage. It might seem that a system should maintain as much descriptive storage as possible, thereby decreasing the processing cost as much as possible. This approach, of course, is naive as it ignores the cost of storing the descriptive information.

To analyze the various costs in a file system which uses descriptive storage we have developed (section 1.3) an idealized model of cost, called the Idealized Descriptive Storage Model (IDSM). Using this model, we will demonstrate the trade-offs between the cost of storage and the cost of processing requests in these systems. We will derive an upper bound for the amount of descriptive storage at which the

expected system cost can be minimum.

It is important that the limited amount of descriptive storage be able to be used for a wide range of processing requests. Since the descriptive storage in many systems can be modelled as a collection of sets we are motivated to define Set-theoretic Descriptive Storage (STDS). Informally, an STDS is a file system in which descriptive storage is composed of sets. Each set has associated with it a definition and is composed (abstractly) of all records in the file for which the definition is true. In the preceding example, the set consisted of all records having the property "student lives in residence". The actual representation of the set might be as a collection of pointers. Several familiar file systems are examples of STDS systems; i.e., inverted files, multilist files, dictionary systems, and a structure proposed by Wong and Chiang [WONG71].

1.2 HISTORICAL OVERVIEW

A review of the history of file systems and their evolution into what are currently called data-base systems indicates several generalizations:

(i) The main volume of the published works is concerned with the exposition of new structures or algorithms.

(ii) Where an analysis to determine the best implementation of a particular file system has appeared, the results are usually based upon restrictive assumptions and a simple definition of cost.

(iii) For particular file systems a retrieval algorithm is usually non-adaptive; i.e., the algorithm consists of a single strategy as opposed to an adaptive method which chooses the best (based upon cost) strategy for an arbitrary query and the descriptive storage defined.

The idea of a data base has evolved from the punched-card and magnetic-tape systems which were developed for first-and second-generation computers. In these first systems the physical order of the records in the files determined the order in which records were encountered by a processing program. The records were ordered according to some unique attribute - leading to the development of unnatural (but unique) identifiers such as student-identification numbers for files containing student records. For many applications, the order in which the records were stored was inconvenient. Thus, sorting techniques became important and a number of methods were developed. The cost of processing in these

systems was largely dependent upon the number of times the file was read. Hence, many requests were collected before the file was processed and the requests were all handled during a single computer run. As the amount of information which a record could potentially contain increased, it became apparent that a fixed format for the records could lead to excessively large records. Hence, the idea of a "formatted record" was developed. Records were "self-identifying", i.e., in addition to the basic information, the records contained information by which a processing program could determine what information was contained in a record.

The usefulness of these systems is exemplified by the number of applications which are still performed using their methodology. The principal disadvantages of this method of operation are as follows:

(i) Batching of requests may cause the time between submission and implementation of a processing request to be excessively long.

(ii) The nature of the hardware prevents the processing of records in any order except the physical order, and all records must be read (or at least up to the last one required) whenever the file is processed.

For many applications, the inexpensiveness of tape or cards as a storage medium (compared to direct-access storage) is still a more important consideration than the disadvantages cited above.

The development of direct-access storage devices,

principally disks and drums, enabled systems designers to implement files in which the two disadvantages above could be avoided. These new techniques included inverted files, multilist files, ring structures, and other list structures. Dodd [DODD69], Lefkovitz [LEFK69], and Morgan [MORG72A] present surveys of these methods and the bibliography associated with them. In general, the results concern the development of new file structures and processing techniques. With the exception of tree structures [SUSS63, CLAMP64, ARORA69, PATT69, BAYER70, KNUTH70, BAYER71, DEM71, NIEV72, NIEV73, WALK72, FOST73] and hashing techniques [MAUR68, MOR68, BLOOM70, COFF70, LUM70, LUM71, KNOTT71, LUM72, BRENT73, LUM73] there is a lack of results concerning the optimization of file structures. By such an optimization, we mean the implementation of a file system so that the processing cost is minimum. In addition to the analytic results cited above, several authors have obtained important results by simulation [ATW72, SEV72]. In general, the results have used exceedingly simple definitions of cost. They assume, for example, that cost is proportional to the number of accesses to direct-access storage. The state of the art is illustrated by a chart [PALM73] published in November 1973. Twenty-one data-base systems were characterized. Under the classification "Tuning and Optimization" the following results were recorded.

Response	Number of Systems
none	10
limited	3
very limited	2
some facilities	3
using operating system facilities	2
some statistics gathered	1

Of the 21 commercially available systems, none claimed to have comprehensive facilities and about half claimed only limited facilities.

The second concept which extends through the literature is a restricted idea of retrieval. Usually, only one retrieval algorithm is presented for a given file system. In addition, the algorithms can usually be applied only to a restricted form of query. An exception to this trend is the work of Hsiao and Harary [HSIA070], and the extension by Manola and Hsiao [MAN73]. These authors have modelled several well-known file structures as special cases of a "generalized file structure". We shall discuss this work in detail in chapter four and we shall present several extensions to the methods of processing this file structure.

Two aspects are apparent with regard to the future of data-base systems:

(i) There will be a large commitment of resources toward the development and usage of large data-base systems.

(ii) It will become increasingly important that these

large data bases have facilities whereby these systems may be structured to operate at or near the minimum cost.

Two sets of reports which are likely to influence the development of these data bases are the CODASYL reports [CODAS69A,CODAS69B,CODAS71] and the SHARE/GUIDE report [GUIDE70]. Both these organizations envision a Data Base Administrator (DBA), or equivalent person, to be responsible, among other functions, for the control of the cost of a data base. As illustrated above, the DBA has relatively few tools to employ in this activity.

1.3 IDSM: An Idealized view of System Cost

As we have noted, one may divide the stored information in many file systems into two classes: descriptive storage and basic storage. Descriptive storage is information whose only inherent value is to facilitate access to basic storage. The motivation for the development of descriptive storage is to provide access to proper subsets of basic storage, thus avoiding the serial examination of all the basic information. For example, in a file consisting of records of persons employed by a company, a set of record addresses, indicating the records of all employers satisfying the criterion "sex is male", could be defined. This set (rather than the entire file) could be used to locate all records of employees with the criterion "sex is male and earns more than $10,000 annually".

In our idealization we shall consider a single computer system in which the amount of direct-access storage (one hierarchy) is sufficient to store the basic and descriptive storage which might be defined. It is assumed that unused quantities of direct-access storage are not charged to the file system being considered. Our point of view is essentially that of a user of a computer utility who is charged only for the resources he explicitly uses.

We shall develop a theoretical model of cost, called the Idealized Descriptive Storage Model (IDSM), to describe situations such as the one outlined above. We shall view the

basic information as being invariant and shall allow the amount of descriptive storage to vary. We define the total cost of such a system to be the sum of two components: manipulative cost and storage cost:

1.3-1 $$T(s) = S(s) + M(s)$$

where s = amount of descriptive storage allocated

T = total-cost function

S = storage-cost function

M = manipulative-cost function.

The storage cost $S(s)$, is the cost to store s units of descriptive storage and to store the basic storage. The manipulative cost is the cost of manipulating the basic information, presumeably the performance of retrievals and updates. All costs are taken over some arbitrary time interval. We shall ignore intangible costs such as those experienced by terminal users of a time-sharing system with a long response time.

Suppose that we decide to allocate s units of descriptive storage. Then, there is a (possibly infinite) number of ways to choose descriptive storage. For each of these ways we will assume that there exists a finite expected manipulative cost, during some time period. Let $M_0(s)$ be the minimum value for these expected manipulation costs, when s units of descriptive storage are allocated. We assume that the cost of storage is an arithmetic progression, depending upon the amount allocated:

1.3-2 $S(s) = B + d \cdot s$

where s = number of units of descriptive storage allocated

B = cost (invariant) to store the basic information

d = cost to store each unit of descriptive storage

Combining the cost of storage with the minimum of expected manipulation cost, we derive the minimum expected cost, $T_0(s)$ to operate the file system for the given period of time, when s units of descriptive storage are allocated. These costs are summarized graphically in diagram 1.3-3. As a convenience, all functions are drawn as continuous curves. Of course, the functions are only defined for integral allocations of descriptive storage.

The following properties are immediately apparent:

1.3-4 property: $M_0(s)$ is a decreasing sequence.

1.3-5 property: $M_0(s)$ approaches a limit Mt,

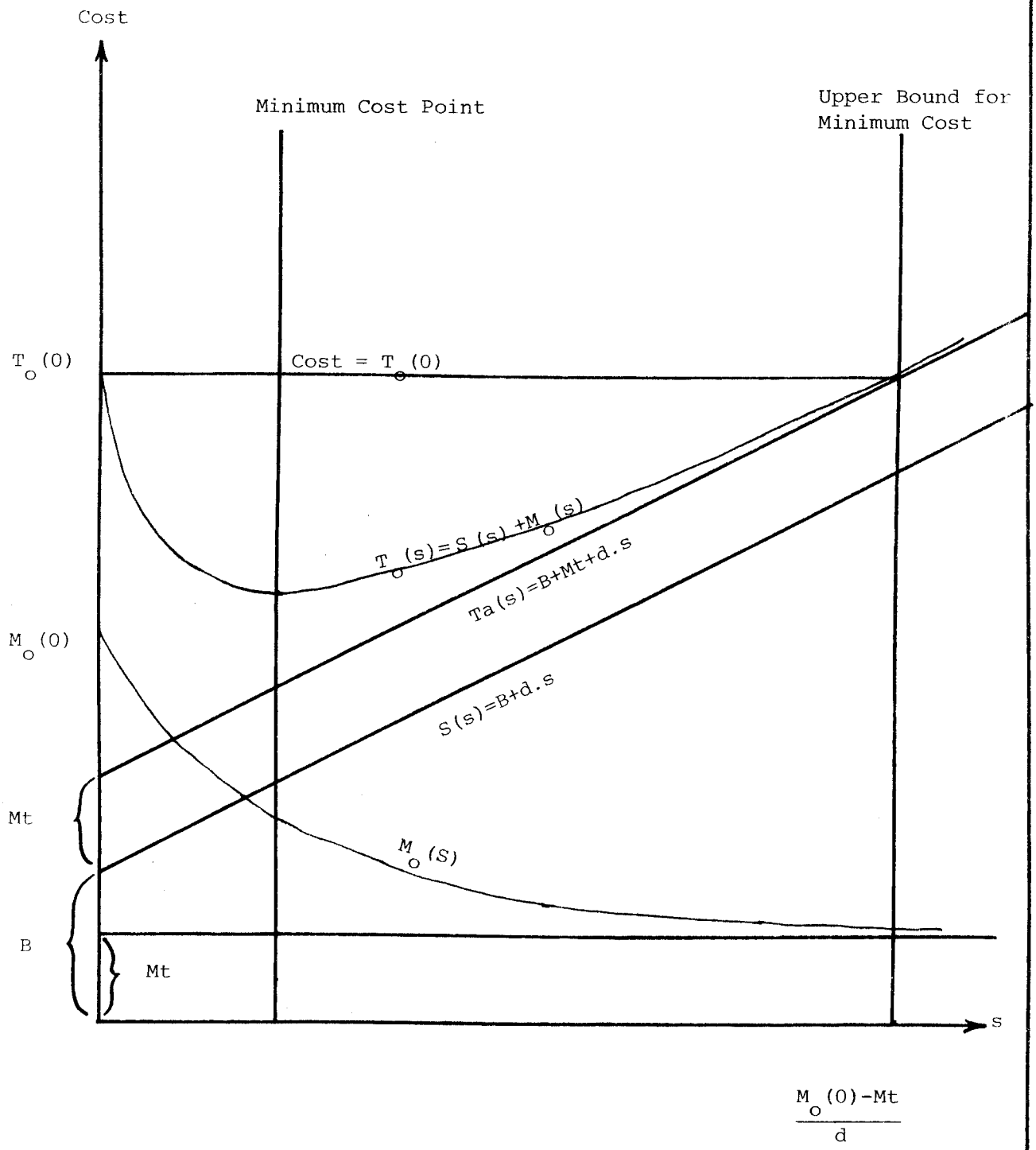
i.e., $\lim_{s \to \infty} M_0(s) = Mt$

1.3-6 property Total cost approaches a sequence $Ta(s)$

where $Ta(s) = B + Mt + d \cdot s$.

Since all descriptive-storage arrangements of s units are also descriptive-storage arrangements of (s+1) units (with one unit unused), it follows that $M_0(s) \geq M_0(s+1)$ and so $M_0(s)$ is a decreasing function. Because $M_0(s)$ is bounded below (by, for example, zero cost) it has a greatest lower bound Mt, called

Diagram 1.3-3: Idealized Descriptive Storage Model



the manipulation threshold. Since $M_0(s)$ is also decreasing, Mt must be the limit of the function as s increases without bound. Property 1.3-6 is a direct consequence of property 1.3-5. As a consequence of the above arguments we conclude

1.3-7 property: Total cost $T_0(s)$ is bounded below by the sequence Ta(s).

We are interested in the amount(s) of descriptive storage for which $T_0(s)$ is minimum, i.e., the amount(s) of descriptive storage for which the expected cost is minimum. We may restrict the points for which $T_0(s)$ is minimum by the following property.

1.3-8 property: $T_0(S)$ is not minimum when

$s > (M_0(0)-Mt)/d$

which may be proven as follows:

1.3-9 proof of property 1.3-8

Let s be any amount of descriptive storage for which T_0 is minimum. Then, $T_0(0) \geq T_0(s)$. By property 1.3-7 we also have $T_0(s) \geq Ta(s)$ and so $T_0(0) \geq Ta(s)$. Hence, we have

$P + M_0(0) \geq B + Mt + d \cdot s$ or

$(M_0(0) - Mt)/d \geq s$

which proves the 1.3-8.

Referring to diagram 1.3-3, we see that a horizontal line cost=$T_0(0)$ intersects the line Ta when $(M_0(0)-Mt)/d = s$. In other words, minimum total cost cannot occur when the lower bound on cost (Ta(s)) exceeds the cost when no descriptive storage is allocated ($T_0(0)$).

We have developed the IDSM to precisely outline the effect of descriptive storage upon cost. The idealized view illustrates the trade-off between decreasing manipulation cost and increasing storage cost, as the amount of descriptive storage is increased. The effect of the cost of storage is characterized by the parameter d (cost to store one unit of descriptive storage). Referring to diagram 1.3-3 we note that a decrease in the value of d has the effects of:

(i) increasing the range of values for which minimum cost may occur.

(ii) decreasing the cost, B, of basic storage

(iii) causing the line S(s) to become more horizontal.

As manipulation cost is not affected by this change, the minimum cost point(s) will be shifted, in diagram 1.3-3, to the right. Thus, our intuition is reinforced: a decrease in the cost of descriptive storage implies that more may be allocated profitably.

Our arguments have been for all arrangements of descriptive storage. Of course, the same arguments could be applied to a particular class of descriptive-storage systems. It is possible, for example, to restrict our view to inverted

files and thereby define $M_0(s)$ to be the minimum cost of an inverted file when s units of descriptive storage may be allocated.

We claim that the IDSM is useful because

(i) It illustrates the trade-off between storage and manipulative costs in a system which uses descriptive storage; and

(ii) It indicates a lower bound for expected system cost, when the amount of descriptive storage is specified; and

(iii) It indicates an upper bound for the amount of descriptive storage for which expected system cost can be minimum.

Its practical value is limited because of the difficulty in determining values for minimum expected manipulation cost, $M_0(s)$, and the greatest lower bound, Mt, for manipulation cost.

We will be concerned with a particular type of file system, Set-theoretical Descriptive Storage (STDS), defined in the following section. Our emphasis will be with regard to retrieval. Thus, we shall consider STDS systems in which manipulation cost is mostly due to retrieval. Within this context, we shall make the following assumption: increasing the amount of descriptive storage will not increase the expected manipulation cost of the system. Thus, we implicitly assume that retrieval algorithms are able to take advantage of the extra descriptive storage without any increase in cost.

In some systems this assumption may not be absolutely valid. For example, the additional descriptive storage may increase the cost of scanning (during retrieval) for useful sets. It is presumed that the assumption is reasonable and will not distort the analyses which follow.

1.4 STDS: A general class of Descriptive Storage

In this section we shall define a general class of descriptive storage, called Set-theoretical Descriptive Storage (STDS). Informally, an STDS is a descriptive storage composed of sets such that:

(i) Each set has a definition in the form of a Boolean expression whose variables are attributes of the data items in basic storage; and

(ii) Each set consists of all those data items in basic storage for which the definition of the set is true.

Before formally defining an STDS we shall consider several preliminary examples of STDS systems.

The examples will refer to the information in table 1.4-1. There are eight rows to the table, where each row represents a person in a mailing-list system. Each person is a member of one or more mailing lists, represented by capital letters following the name (i.e., person P5 is on mailing lists B and C). In each of the examples which follow, we shall suppose that basic storage consists of eight data items, where each data item contains:

(i) the person's name

(ii) the mailing lists applying to the person

(iii) the mailing address(es) of the person

(iv) his title

(v) his occupation

As we will be concerned with only the first two elements in our examples, table 1.4-1 shows only persons' names and the

mailing lists which apply. We shall illustrate several well-known STDS systems by constructing descriptive storage for this example.

We shall first consider an inverted file. Table 1.4-2 gives a descriptive storage for this case. In an inverted file some (or all) of the attributes in basic storage are selected. A set is maintained for each of these attributes and consists of all data items for which the attribute is true. When all the attributes are chosen, the system is said to be fully inverted. To construct the three sets illustrated in table 1.4-2 we have chosen, as attributes, whether or not a person is on a particular mailing list. Historically, basic storage has been implemented as a file of records and the sets, called inverted lists, have been represented as a collection of (hardware) record addresses, record keys, or accession numbers. Alternatively, the inverted lists have been constructed as binary sequences where the i-th digit in the j-th sequence is one if and only if the j-th attribute is true for the i-th record. We note that descriptive storage may be logically thought to consist of sets of data items and that each set consists of all data items which satisfy the associated definition.

A second example is what we term a Wong-and-Chiang (W&C) implementation [WONG71] of descriptive storage. Table 1.4-3 illustrates the descriptive storage in a W&C system, when the attributes are identical to those in the first example. Treating each attribute as a bivalent variable, we may

Table 1.4-1 Persons in file and mailing lists.

P1 : A

P2 : B

P3 : A,B

P4 : C

P5 : B,C

P6 : A

P7 : A

P8 : C

Table 1.4-2 Inverted Lists for table 1.4-1

A : {P1,P3,P6,P7}

B : {P2,P3,P5}

C : {P4,P5,P8}

Table 1.4-3 Wong-and-Chiang atoms and sets for table 1.4-1

ATOM		SET
$A \wedge \bar{B} \wedge \bar{C}$	:	{P1,P6,P7}
$\bar{A} \wedge B \wedge \bar{C}$	:	{P2}
$A \wedge B \wedge \bar{C}$	:	{P3}
$\bar{A} \wedge \bar{B} \wedge C$	:	{P4,P8}
$\bar{A} \wedge B \wedge C$	:	{P5}

construct Boolean expressions, in the usual way. Writing these expressions in disjunctive normal form, there are 2^n minimal expressions (called atoms) for n attributes. Each expression may be expressed as the conjunction of atoms. Wong and Chiang propose that a set be constructed for every atom, consisting of those data items for which the atom is true. Of course, atoms to which null sets correspond would be disregarded in an actual implementation. It is shown that every data item occurs in exactly one set. The collection of records for which an expression is true may be found by taking the union of all the sets corresponding to the atoms whose conjunction is the expression. In table 1.4-3, we note that these are five (out of a possible eight) atoms with non-null sets and that every data item is found in exactly one set.

The similarities in the two examples motivate our definition of Set-theoretical Descriptive Storage (STDS). We recall that in both cases descriptive storage was composed of sets. Each of these sets has a definition and the content of each set is those data items in basic storage for which the associated definition is true. We will define the sets in descriptive storage using a function Δ which determines, for a Boolean expression e, that set which consists of exactly those data items for which e is true. As a notational convenience, we shall represent by D(S) the Boolean expression used to generate S. For consistency, we will permit in an STDS only those sets consisting of exactly those data items in basic storage for which the associated definition is true.

Formally, we define an STDS as follows.

1.4-4 Definition: An STDS is a four-tuple $\langle I, A, D, \mathcal{S} \rangle$ where

(i) I (information) is a finite collection of undefined elements called data items or records.

(ii) A is a finite set of attributes, each of which may be evaluated as either true (1) or false (0) for every data item $d \in I$. For $d \in I$ and $a \in A$, we denote this evaluation as $E(d,a)$.

(iii) Δ: $\beta(A) \rightarrow \top\top(I)$ is a function which defines, for any of the expressions ($\beta(A)$) having attributes from A as variables, a member of the power set of I ($\top\top(I)$) as follows. For any $e \in \beta(A)$,

$$\Delta(e) = \{d \mid (d \in I)(E(d,e)=1)\}$$

where $E(d,e)$ is the result of evaluating the expression e when all the variables $a \in A$ in e are valuated as $E(d,a)$.

(iv) $\mathcal{S} \subseteq \{\Delta(e) \mid e \in \beta(A)\}$. For a set $S \in \mathcal{S}$, we denote by $D(S)$ the boolean expression which generated S, and so we can write

$$S = \{d \mid (d \in I)(E(d,D(S))=1)\}.$$

In other words, an STDS has descriptive storage composed of subsets of basic storage. Each subset has a definition and consists of all items in basic storage for which the definition is true. It may be verified that inverted files and WEC files are instances of this general definition. In

addition, we shall demonstrate that multilist files and dictionary systems are special cases of an STDS.

A multilist file is similar to an inverted file. In both cases, a collection of attributes is selected and a set is constructed for each of these attributes. Each set consists of all data items for which the corresponding attribute is true. The difference between the two types of systems is in the way they are implemented. We recall that the sets, in inverted files, were constructed as distinct entities, separate from the file and from each other. Sets are implemented in a multilist system by using pointers which are associated with the records of the original file. Pointers are used to uniquely identify records in basic storage. Examples of pointers include

(i) the hardware address of a record

(ii) the relative record number of a record.

A multilist set is implemented as a sequence of data items $\{r_1, r_2, \ldots, r_k\}$ where

(i) For each of the chosen attributes, there is a special pointer identifying the first record (r_1) in the sequence

(ii) Record r_i ($1 \leq i < k$) contains a pointer which identifies the next record (r_{i+1}) in the sequence.

(iii) The last record (r_k) in the sequence contains a special pointer, called a null pointer, indicating the end of the sequence.

These sets are constructed so as to contain all data items for

which the corresponding attribute is true. Hence, the multilist file is an example of an STDS. Methods of retrieval using a multilist file are referenced in [PRYW63, HSIA070, MAN73] and will be discussed in section 4.2.2. Table 1.4-5 is a schematic representation of a multilist file for the data in table 1.4-1. A multilist set is constructed for each of the three mailing lists. Pointers are represented as ordered pairs where the first element is the mailing list and the second element is the name of the person for the next record in the set; i.e., <A,P1> is a pointer for mailing list A and indicates the record for person P1. The special symbol 'Ø' is used for the null pointer.

A fourth example of an STDS is a dictionary system. In this situation, records are located by using a unique identifier, called a key, associated with each record. The dictionary is descriptive storage which contains all keys and a pointer associated with each key. A record is located by searching the dictionary for the key of the record and the associated pointer of this key indicates the location of the record. We consider each key to be an attribute. Thus, for every attribute there exists a set consisting of a single data item and this data item is the only one having that attribute. Hence, this case is an example of an STDS. To facilitate searching for keys it is common to establish a hierarchy as follows:

(i) Order the keys according to some collating sequence.

Table 1.4.5 Example of Descriptive Storage For Multilist File

(i) Pointers to First Record in Each Sequence

<A,P1>

<B,P2>

<C,P4>

(ii) Representation of the File

P1 : <A,P3>

P2 : <B,P3>

P3 : <A,P6>, <B,P5>

P4 : <C,P5>

P5 : <B,Ø>, <C,P8>

P6 : <A,P7>

P7 : <A,Ø>

P8 : <C,Ø>

notes: (1) <K,P> represents K-pointer to the record of person "P".

(2) <K,Ø> represents a null K-pointer.

(ii) Partition the keys into sub-tables of some maximum size.

(iii) Establish a key for each subtable and associate with this key a pointer to the subtable.

(iv) This establishes a level of the table. Repeat at (i), using the keys from (iii), until only one sub-table remains.

Searching for a key involves one access to each level of the table, until the highest level is determined and the required entry found. As an example, table 1.4-7 illustrates a dictionary where:

(i) the keys are the persons' names of table 1.4-1

(ii) the maximum sub-table size is two entries

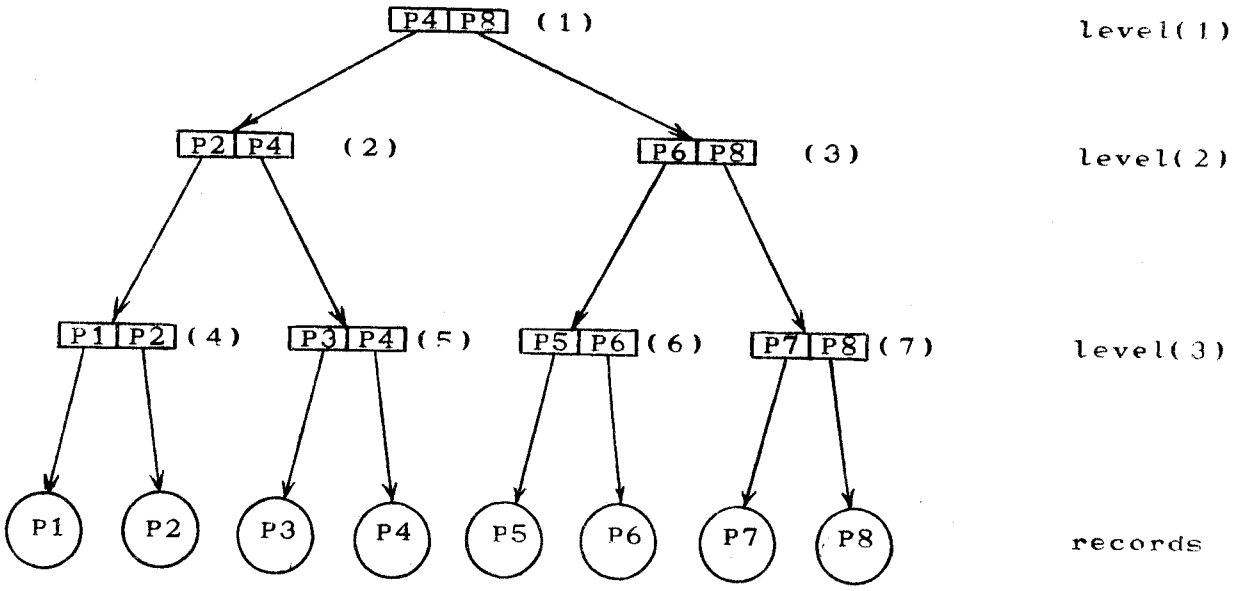
(iii) the key of a subtable is the 'highest' key alphabetically

(iv) each subtable is entirely used.

We represent pointers by arrows. Thus to locate the record with key 'P5', references are made to subtables 1, 3, and 6 respectively. We claim that the situation outlined is another example of an STDS. Each entry in the dictionary defines a set of data items having one of the attributes of the records to which it points, directly or indirectly. Thus, the entry for 'P2' in subtable (2) of diagram 1.4-7 indicates data items with keys 'P1' or 'P2'.

We have indicated that several well-known file systems are examples of the STDS definition (1.4-4). The literature

Diagram 1.4-7: Example of dictionary system



applying to these particular systems is voluminous and will be referenced in Chapter Four. Our general point-of-view introduces a number of questions, i.e.,

(i) Is it possible to implement, in full generality, an STDS system?

(ii) Given that (i) is possible, how may we determine the best STDS for a given purpose?

(iii) What improvements to particular STDS systems are indicated by considering the general case?

The following chapters will outline our answers to the above questions.

CHAPTER 2: STDS IMPLEMENTATION: A GENERAL ANALYSIS

2.1 INTRODUCTION

In this chapter we shall consider Set-theoretical Descriptive Storage (STDS) from a general point of view. We recall the definition of STDS as a four-tuple $\langle I,A,D,\mathcal{S}\rangle$ where I is a finite set of data items, A is a collection of attributes having a value of true (1) or false (0) for every data item in I, and $\mathcal{S}$ is a finite collection of subsets of I where for every $S\in\mathcal{S}$, $S=\{d|(d\in I)(E(d,D(S))=1)\}$

The major problem analyzed in this chapter is how to retrieve using STDS. For a given Boolean expression of attributes, we would like to determine how to combine some or all of the sets in descriptive storage in order to derive the collection of records for which this query is true. We have developed the theory of functional-covering (f-covers) to specify how this combination may be implemented.

Trivially, if a query is identical to the definition of one of the sets in descriptive storage, then that set is the collection of data items for which the query is true. In other cases, several sets in descriptive storage may be combined, using the set-theoretical operations of union, intersection, and complementation to derive the required collection. It is possible that no combination will yield exactly the collection of data items for which the query is true. In this situation, a possible solution is to obtain a superset of the required

collection. We have developed the theory of f-covers to specify how to use the set definitions in order to determine the smallest superset that contains the data items for which the query is true. We develop the theory further in order to determine supersets which may be constructed with any of the following restrictions applied to them:

2.1-1: (i) Complementation of sets is not allowed in the construction of the superset.

(ii) Union of sets is not permitted.

(iii) Intersection of sets is not permitted.

or (iv) We allow only union of sets or only intersection of sets in order to construct the f-cover.

Although the main thrust of this chapter is towards retrieval, we do not consider updates to be inconsistent for an STDS. In section 2.3 we will discuss the problem of updating an STDS.

For any system to be successful, it must operate with a low enough cost to satisfy the users of it. In section 2.4 we will analyze the costs in an STDS system. The total cost of a system may change over a period of time due to, for example, changed usage characteristics. As a result, the system must be monitored to verify that its operation is reasonably close to the predicted situation. When the system characteristics have changed sufficiently that a restructuring or optimization is indicated, the new system definition must be presented to the optimization procedure. Thus we propose a

model of system operation (section 2.5) called the feedback model. In this model the file system is monitored and restructured when necessary. This restructuring activity is the subject of Chapter 3.

In summary, the main purposes of this chapter are:

(i) to demonstrate that STDS is a useful and feasible model with regard to information retrieval.

(ii) to analyze the costs of such a system

(iii) to develop a general framework within which the optimization procedures of Chapter 3 are performed.

2.2 RETRIEVAL USING STDS

In this section we shall demonstrate how to retrieve using STDS. A general technique will be developed which involves creating a set of data items containing those items requested by a Boolean expression called a query. Unacceptable data items are then eliminated from this larger set. A systematic method for creating such supersets, based upon the theory in Appendix C, is explained. This method entails creating a special Boolean expression called an f-cover which is true whenever the query is true. The composition of the derived f-cover specifies how to combine sets of descriptive storage to create the required superset. We will show how to create a number of different f-covers and we indicate the advantages of each form. Several remarks will be made concerning how to choose the appropriate f-cover for an expression.

By a **retrieval**, we mean a process which determines, for a given Boolean expression q called a **query**, a retrieval set R(q) which satisfies

2.2-1 $$R(q) = \{d \mid (d \in I)(E(d,q)=1)\}$$

Recalling that the notation E(d,q) indicates whether or not a data item $d \in I$ is true for an expression q, we see that the retrieval process selects all data items $d \in I$ for which the query is true.

This definition of retrieval suggests the following approach.

(i) Create a set which contains the retrieval set.

(ii) Eliminate from this superset all data items not in the retrieval set.

We note that one set which always contains the retrieval set is the set I of all data items. When this set is used as the superset, then retrieval is accomplished in a manner similar to the traditional method of sequentially processing files, i.e., examine all the data items, in some order, selecting those for which the query is true. We shall assume that we can always use the set I as a superset and shall call this assumption the sequential assumption.

In many situations, however, the sets in descriptive storage can be of use. For example, it may be the case that a set S has the definition D(S)=q and so the required retrieval set would be S (and step ii would be unnecessary). In other situations, two or more sets from descriptive storage may be combined in some fashion to derive the superset.

We have developed the theory of functional covering (f-covering) as the basis for the methods which determine these supersets. Appendix C is a formal presentation of this theory. Within this section we shall be concerned with only the results and application of this theory. First, we present some preliminary definitions and lemmas.

2.2-2 Definition: A Boolean expression f implies a Boolean expression g, if for all valuations of the variables in expression f for which f is true, it follows that g is true. In this case we also say

that g <u>covers</u> f, written f==>g, and that g is a <u>cover</u> of f.

As a consequence of this definition, the following lemmas are easily proven. For the Boolean expressions f, f_1, f_2, g, g_1, and g_2 we have:

<u>2.2-3</u> <u>lemma</u>: If f_1 ==> g_1, and f_2 ==> g_2, then

(i) $f_1 \wedge f_2$ ==> $g_1 \wedge g_2$

(ii) $f_1 \vee f_2$ ==> $g_1 \vee g_2$

<u>2.2-4</u> <u>lemma</u>: If f ==> g, then $\bar{g}$ ==> $\bar{f}$.

<u>2.2-5</u> <u>lemma</u>: f ==> $g_1 \wedge g_2$ if and only if f ==> g_1 and f ==> g_2.

<u>2.2-6</u> <u>lemma</u>: $f_1 \vee f_2$ ==> g if and only if f_1 ==> g and f_2 ==> g.

A generalization of a cover is an f-cover:

<u>2.2-7</u> <u>Definition</u>: For a set of Boolean expressions $\{f_1, f_2, \ldots, f_n\}$, an <u>f-cover</u> of a Boolean expression q is an expression $C(f_1,f_2,\ldots,f_n)$ for which q ==> C.

In other words, an f-cover of a Boolean expression q is a Boolean expression which has f_1, f_2, ..., f_n as variables and which covers q.

Consider the following example (reproduced in section C.3). Let

2.2-8 $q = (a \vee b) \wedge d$

$f_1 = a$

$f_2 = b$

$f_3 = a \wedge c$

Then, $f_1 \vee f_2 \bar{f}_3$ and $f_1 \vee f_2$ are two examples of f-covers.

An f-cover which has special importance is a minimal f-cover.

2.2-9 Definition: For the set of Boolean expressions $\{f_1, f_2, \ldots, f_n\}$, an f-cover C of an expression q is a minimal f-cover if for any other f-cover C' of q, C==>C'.

Thus, a minimal f-cover is covered by all other f-covers. Referring to example 2.2-8, $f_1 \vee f_2 \bar{f}_3$ and $f_1 \vee f_2$ are both (equivalent) minimal f-covers. Appendix C outlines a method for constructing minimal f-covers.

We now specify how f-covers can be used in the retrieval process. We define a retrieval algorithm as follows:

2.2-10 (i) For the query expresssion q and the set of Boolean expressions, $\{D(S) | (S \in \$)\}$, construct an f-cover C.

(ii) Use C, together with the contents of sets $S \in \$$ to create a superset R(C) of the retrieval set.

(iii) Select (eliminate) data items $d \in R(C)$ for which the expression q is true (false) to create

the retrieval set R(q).

Before considering how to accomplish step (ii) of 2.2-10 we present the following lemma (proven in Appendix C).

2.2-11 **Lemma**: Let S_1, $S_2 \in \mathcal{S}$ be defined in an STDS $\langle I, A, D, \mathcal{S} \rangle$. Then,

(i) $S_1 \cap S_2 = \{d \mid (d \in I)(E(d, D(S_1) \wedge D(S_2)) = 1)\}$

(ii) $S_1 \cup S_2 = \{d \mid (d \in I)(E(d, D(S_1) \vee D(S_2)) = 1)\}$

(iii) $\bar{S}_1 = \{d \mid (d \in I)(d \notin S_1)\}$ (by definition)

$= \{d \mid (d \in I)(E(d, \overline{D(S_1)}) = 1)\}$

Hence, for an STDS $\langle I, A, D, \mathcal{S} \rangle$ and a Boolean expression q, let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ and let $C(D(S_1), D(S_2), \ldots, D(S_n))$ be an f-cover for q. Then, we may construct the superset

2.2-12 $R(C) = \{d \mid (d \in I)(E(d, C) = 1)\}$

by taking the expression for C and making the following (simultaneous) substitutions:

2.2-13 (i) S_i for $D(S_i)$, $i = 1, 2, \ldots, k$

(ii) '$\cap$' for '$\wedge$'

(iii) '$\cup$' for '$\vee$'

Then, for an f-cover $C = D(S_1) \wedge \overline{D(S_2)}$, the corresponding set is $R(C) = S_1 \cap \bar{S}_2 = \{d \mid (d \in I)(E(d, D(S_1) \wedge \overline{D(S_2)}) = 1)\}$.

In summary, we have developed a general method for determining the set of data items in basic storage for which an arbitrary query expression is true. This method involves

choosing an appropriate f-cover, creating the corresponding superset of data items, and finally selecting (eliminating) the acceptable (unacceptable) data items. The first stage of the method involves manipulation of the set definitions, disregarding the set contents. In the second stage the set contents, in conjunction with an expression derived in the first stage, are used. It is probable that the definitions and contents would be stored in direct-access storage media and transferred to the memory of a computer when required. If the definition and contents were physically represented such that the transfer of one implied the transfer of the other, then more information than is necessary is likely to be transferred. Thus, it is indicated that the definitions and contents be physically separated in the storage media. We call this generalization the separation principle.

We have been primarily concerned with minimal f-covers because the number of data items corresponding to such an f-cover is minimum. This is a consequence of the following lemma and its corollary.

2.2-14 lemma: In the STDS $\langle I,C,D,\mathcal{S}\rangle$ where S_1, $S_2 \in \mathcal{S}$, if $D(S_1) \Longrightarrow D(S_2)$, then $S_1 \subseteq S_2$.

2.2-15 corollary: In the STDS $\langle I,A,D,\mathcal{S}\rangle$ let q be an expression depending upon the variables in A. Then the number of data items in the set corresponding to a minimal f-cover of q is minimum.

Despite the fact that a minimal f-cover corresponds to the smallest number of data items of all f-covers, it may not be the best choice for a particular STDS. This is because we have ignored the costs of constructing the minimal f-cover and of constructing the set of data items corresponding to it. For example, consider the case where I is a file and the data items are records located at unpredictable addresses in a direct-access storage medium. If the contents of sets were implemented as collections of these addresses, then the process of complementing a set would likely be very costly. A less expensive alternative might be to construct a (possibly non-minimal) f-cover in which complementation is not used. In other situations, reasons (i.e., programming simplicity) may warrant that f-covers which use only 'AND' or 'OR' operations be used. For these considerations, we have indicated (in Appendix C) how to construct (if they exist) f-covers with any of the following restrictions:

2.2-16 (i) No complementation is allowed in the f-cover expression.

(ii) 'AND' operations are not permitted

(iii) 'OR' operations are not permitted

or (iv) Only expressions with condition (i) and either of (ii) or (iii) are allowed.

Thus, we can construct particular f-covers which have a given (restricted) composition if they exist. In some cases, a number of such f-covers is constructed. In this case, some

criteria must be established to decide which f-cover to use. This choice could be either arbitrary (i.e., take the first) or could be based upon some estimate of cost.

As an example, suppose we restrict f-covers with conditions (i) and (ii) of 2.2-16. In other words, for an STDS $\langle I, A, D, \mathcal{S} \rangle$ in which $S = \{S_1, S_2, \ldots, S_n\}$, we allow only f-covers of the form:

2.2-17 $$U = \bigvee_{i \in Q} D(S_i) \text{ where } Q \subseteq \{1, 2, \ldots, K\}.$$

Let us suppose that the number of data items in a set $S \in \mathcal{S}$ is given by $l(S)$ and let us suppose that the number of data items in set I is given by the integer t. Then, for an f-cover of the form 2.2-17, we may compute

2.2-18 $l(S_i)/t$ is an estimate of the probability that a data item occurs in set S_i, for $i \in Q$.

2.2-19 $(1 - l(S_i)/t)$ is an estimate of the probability that a data item does not occur in S_i, for $i \in Q$.

2.2-20: $\prod_{i \in Q} (1 - l(S_i)/t)$ is an estimate of the probability that a data item $d \in I$ does not occur in the set corresponding to $U = \bigcup_{i \in Q} S_i$.

Hence, we can calculate

2.2-21 $$1 - \prod_{i \in Q} (1 - l(S_i)/t)$$

as an estimate of the probability that an arbitrary data item occurs in the set R(U) for the f-cover U. If a number of such f-covers exist, then a selection criteria might be to choose

the f-cover for which the value of 2.2-21 was minimum. An alternative could be to use the value of the expression

2.2-22 $$\sum_{i \in Q} l(S_i)$$

(maximum number of data items corresponding to f-cover) to choose the required f-cover. Similar probabilistic analysis could be used as a selection criteria when other restricted f-covers are derived.

In other situations the cost of retrieving, for a given f-cover, may be estimated. Suppose, for example, that f-covers are restricted as in 2.2-17 and that the superset R(U) is obtained by merging the sorted (according to some appropriate sorting criterion) strings of data items (or their addresses) in S_i, $i \in Q$. Then, an appropriate analysis of cost might be to estimate the cost of the merge and the cost of the final selection (elimination) stage. The f-cover for which the total cost was minimum would then be used for retrieval. Factors to be considered include the algorithms to be used and the accounting parameters and procedures currently employed.

Of course, there is a cost involved in performing the analyses proposed above. This cost must be balanced by a saving during the last two stages of retrieval. It is impossible to predict how complex an analysis should be. It is anticipated, for example, that some systems would perform best with a very simple or even arbitrary selection criterion.

Lastly, we mention that a particular f-cover may be expressed in many forms. For example, the expression $f_1 \vee f_2 \vee f_3$ could be expressed in disjunctive normal form having

seven terms. If $f_1 \Longrightarrow f_2$, then an equivalent expression is $f_2 \vee f_3$. Simpler forms have a similarly simple set-theoretic expression for the set of data items which corresponds to it. This set is easier (and less costly) to construct using a simpler expression. Hence, it may be important to use reduction procedures. Section 5 of Appendix C discusses various methods which may be used to perform these simplications.

In conclusion, we have proposed and justified a general method of retrieval in file systems which may be classified as STDS systems. As we shall see when we analyze particular systems in chapter 4, this general approach suggests several improvements to the manner in which retrieval has been historically implemented.

2.3 UPDATE CONSIDERATIONS

The two main operations in a file system may be summarized as retrieval and update. We considered the former in the preceding section. This section will be concerned with with the update operation. We have intended the STDS model to be primarily used for retrieval. We thus expect that most of the manipulation cost to be attributable to retrieval. We do not, however, consider updates to be inconsistent with the model.

Updates are modifications to basic storage. These changes include adding new data items, deleting existing data items, and modifying the attributes of existing data items. Some of these changes may affect the integrity of descriptive storage unless it too is updated. Usually this loss of integrity cannot be tolerated although a few exceptions (i.e., a few document retrieval systems) to this generalization are plausible.

As an example, consider an STDS where I is a personnel file for a company. The data items in I are records, each of which contains information about a particular employee. One attribute might concern whether or not an employee worked in the finance department. Suppose that a set S with definition "employee works in finance department" exists. When the finance department hires a new employee and a corresponding record is added to the file, S is not valid unless we add the record in question to S. Many such sets may exist and some may require similar additions.

In general, as descriptive storage becomes more elaborate, the process of updating descriptive storage becomes more complex. Whenever a data item is added, deleted, or changed, the data item must be either added to or deleted from the relevent sets in descriptive storage. In some cases, the set definitions must be scanned to determine the sets affected. This processing of the set definitions, disregarding the set contents, is a second argument for the separation principle. Thus, useless transfer of sets' contents from direct-access storage to computer memory is avoided.

An alternative to updating the descriptive storage is to maintain an update area. All changes to basic storage would be recorded in this area. Periodically descriptive storage would be reconstructed to incorporate these changes. The retrieval algorithm, after creating a retrieval set using the original descriptive storage, would be required to add and delete data items according to the information in the update area. Balancing the simplicity of this approach is the complication of the retrieval algorithm(s) and the deterioration in performance as the update area becomes larger. A model of this deterioration is found in [SHN73].

For a particular STDS, a compromise between maintaining an update area and modifying all descriptive storage may be the best solution. For example, deletions from a set may be accomplished by marking a data item in a set as deleted. An addition to a set may be implemented as a "set definition"-

"data item" couple in the update area. In any case, we submit that the STDS structure is a feasible representation to be updated. The complexity of the update mechanism is countered by the gains expected from the generality and ease of retrieval.

2.4 SYSTEM COST

In this section we shall identify the components of cost in an STDS system. Recalling the Idealized Descriptive Storage Model (IDSM) we note that total system cost for a period of time is the sum of storage cost and of manipulation cost.

We shall only be concerned with tangible costs; i.e., those costs which can be determined by standard measurement of computer resources. We shall disregard intangible costs such as those incurred by the users of an on-line computer system in which there is poor response. In computer systems tangible costs are usually determined by special programs and/or subroutines known as accounting procedures. These procedures measure the usage of system resources and compute a cost using a formula in which these measurements are combined. Typical measurements include

(i) the number of cycles of central processor time used by a program

(ii) the number of input/output operations during the execution of a program

(iii) the amount of computer memory dedicated to a program and the time for which that computer memory was used.

(iv) the amount of direct-access storage reserved by a user and the duration of that reservation.

2.4-1 Accounting Formula used at the University of Waterloo for computer jobs run on the IBM 360/75 computer.

Usage (in units) = 700 x CPU + .5 x RW x ERT

+ .0016 x PRT + .0066 x PCH + .0014 x RD

where:

CPU = hours of central processor time used by the program

R = number of kilobytes of memory dedicated to the program

RW = a weighted measure of memory usage

$= .833 \times (R + .002 \times R^2)$

ERT = a measure of real time

= CPU + .027 x IO

IO = number of input/output operations performed by the program

PRT = number of print records

PCH = number of cards punched

RD = number of cards read

* The cost of a unit is approximately $0.10 for a University of Waterloo user.

As an example, we present (table 2.4-1) the accounting formula used at the University of Waterloo Computer Centre to calculate the cost of running a batch program on an IBM/360 Model 75 Computer.

Our concern is primarily with retrieval using the STDS model. Our analyses shall ignore the cost of updating an STDS system. Thus, manipulation cost is the sum of the costs of performing retrievals for the queries introduced into the system. Total cost for an interval of time is therefore dependent upon:

2.4-2

(i) accounting procedures and parameters

(ii) algorithms for retrieval

(iii) nature of the queries

(iv) frequency by which queries are introduced

(v) the definition and size of descriptive storage and of basic storage.

Our goal is to implement an STDS for which this total cost is minimum. We shall assume that (i) and (ii) of 2.4-2 are fixed. By measurement we shall determine (iii) and (iv) of 2.4-2. We shall assume in our analysis that the representation and definition of basic storage is unalterable. Using the above characterization we shall construct a descriptive storage for which the total cost is minimum or nearly so.

This analysis suggests a method of system operation which is the subject of the following section.

2.5 FEEDBACK MODEL

In this chapter we have demonstrated the feasibility of the STDS concept with regard to retrieval and we discussed the costs of operating such a system. The next problem which we will consider is how to choose the best descriptive storage for an STDS system. In general, this is a continuing process because it is not always possible to predict 'a priori' how the basic information will be used. Even if such a prediction could be made, the use of the data could change after a period of time.

These considerations suggest a method of operation which we call the Feedback Model. This model, schematically represented in Diagram 2.5-2, has two processes called the retrieval process and the optimization process. The retrieval process is simply the implementation of the retrieval algorithms described previously, augmented with a measurement facility. The optimization process produces a descriptive storage arrangement based upon:

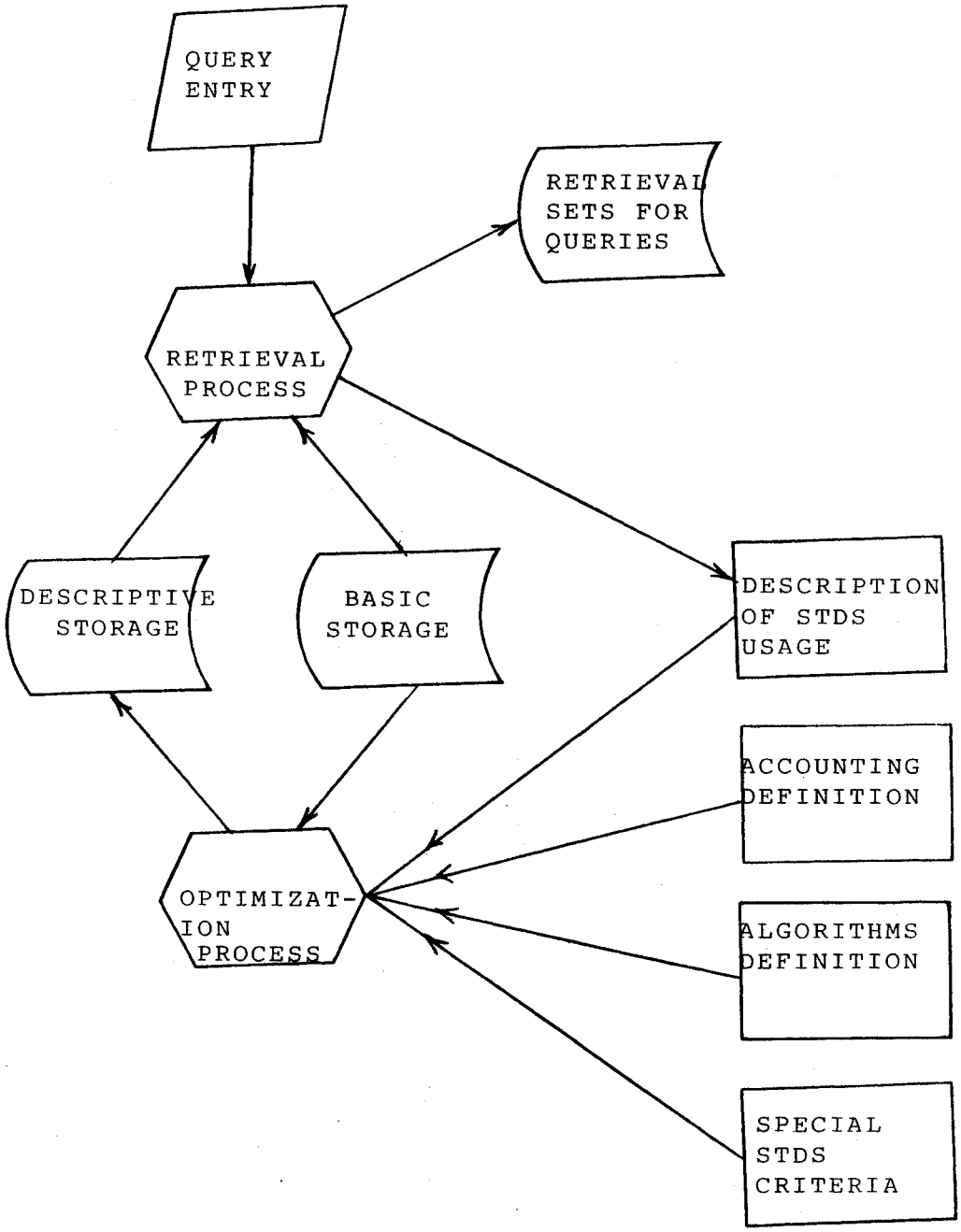
2.5-1: (i) the data in basic storage;

(ii) a description of the retrievals to be performed; (iii) a description of the accounting procedures;

(iv) a description of the retrieval algorithms which may be employed; and

(v) any special STDS criteria (i.e., resultant structure must be a multilist file) the system administrators wish to impose.

Diagram 2.5-2: Feedback model for STDS.



Queries are introduced into the system by some unspecified mechanism. For each query the retrieval process creates a retrieval set consisting of all those data items for which the query is true. The retrieval procedure processes descriptive and basic storages to produce this retrieval set. A measurement apparatus is embedded within the retrieval process so that:

2.5-3: (i) the administrators of the STDS system can determine when the system is no longer functioning as predicted; and

(ii) sufficient information is collected to describe the current usage of the system.

It is not anticipated that the proposed measurement process will be difficult to implement. Most computer systems have accounting procedures which measure usage of computer resources so that the cost of performing each retrieval can be calculated. We also wish to characterize the retrievals that the retrieval process performed. The easiest manner to create this description is to collect the queries which are introduced into the system. We can thus, for example, envision a measurement process which produces for each query an accounting record consisting of:

2.5-4: (i) the query

(ii) the number of cycles of central processor time used to create the retrieval set.

(iii) the number of accesses to descriptive storage

(iv) the number of accesses to basic storage.

All items but the first item above could be used to calculate the cost of performing the query.

The optimization procedure produces descriptive storage based upon the five criteria cited (2.5-1) above. The particular arrangement created is one for which the expected cost of operation will be minimum. As a result the optimization procedure must take into consideration the algorithms used and the method of calculating cost for them. The data items in basic storage determine, for example, the number of items in a set with a given definition and so must be considered. A description of how the system is expected to be used is necessary to compute expected cost for any potential storage arrangement. Lastly, it may have been decided that the STDS must be of a special format (such as an inverted file) and so these special STDS descriptions must be considered.

It is anticipated that a general optimization process is too complicated to construct and/or too expensive to use. Hence, it is proposed that optimization procedures be constructed for each set of special STDS criteria. Chapter 3 is an investigation of this problem in general.

2.6 SUMMARY

This chapter has been concerned with the general feasibility of the STDS model. Using the theory of functional covering (f-covering) we have demonstrated that retrieval is able to be accomplished in a number of workable ways. We have remarked that updates are not inconsistent with the model. It is thus concluded that the STDS model is practical.

We have also proposed a general framework for the operation of an STDS system. The cost components in such an operation have been isolated. The method of operation, called the Feedback Model, specifies two processes, called the retrieval process and the optimization process. We have already, in general, considered the retrieval process. The following chapter will be concerned with the optimization process.

Chapter 3: Optimization of STDS

3.1.1 Introduction

The purpose of this chapter is to discuss how to choose the sets for an STDS system. By optimizing an STDS system, we mean determining a collection of sets so that the expected cost of operating the STDS is near the minimum-cost point indicated by the IDSM (see diagram 1.3-3). We will consider (sub-section 3.1.2) a number of traditional approaches to this problem and conclude that they cannot be applied in general. For situations where the sets cannot be selected using one of these traditional approaches, we suggest that the well-known branch-and-bound (B&B) method be adapted to determine the collection of sets for descriptive storage.

The proposed method of optimization will have two phases:

(i) Generate a collection of sets (candidates) which might be included in descriptive storage.

(ii) Use the branch-and-bound method to determine the best sub-collection from the candidate collection.

In section 3.2.1 we will discuss how to implement the B&B method for this particular problem. The maximum number of steps in this algorithm indicates that this worst case must be avoided. We will specify some heuristic approaches which may be applied to avoid the excessive computation. We will develop a method by which it may be possible (depending upon the

particular optimization) to reduce the optimization to a number of small optimizations. Not only does this approach appear to reduce the amount of computation but it substantially reduces the number of steps of the worst case.

The maximum number of steps in the B&B algorithm is dependent upon the characterization of how the system will be used and upon the number of candidates selected for optimization. In section 3.2.2 we will discuss these two factors. Our prediction of how the system will operate is based upon the historical operation of the system. Associated with each of the queries is its expected frequency of occurance. In some systems it may be desirable to limit the number of queries used in the B&B algorithm. Hence we will propose three methods to reduce the number of queries.

We will show how to generate candidates for descriptive storage using the queries which characterize system usage. We will suggest some restrictions that may be placed upon the candidates so that all possible sets are not considered. We will propose a new application of classification theory. The theory may be used to classify "similar" queries into groups or clusters. Each of these clusters can be used to generate candidates. An optimal collection can be derived for each of these groups of candidates. The union of these optimal groups then can be used as descriptive storage. This method has significant computational advantages.

3.1.2 Motivation for using Branch-and-Bound method

There are various approaches to choosing an STDS arrangement which is nearly optimal. With regard to one of these methods (the analytic approach), we will show that it is always possible to derive a cost function which might be minimized. These methods involve estimating how the system will be used in the future in order to predict expected cost. Such predictions can be based upon the historical operation of the system. We shall determine such a characterization by using the queries which have been posed to the system. Specifically, we shall predict system usage by using two sets, F and Q, each having k elements. The set Q consists of queries and the set F is a collection of positive real numbers such that $f_i \in F$ $(1 \leq i \leq k)$ is an estimate of the frequency that query $q_i \in Q$ is introduced into the system, during some time interval. Section 3.2.2 analyzes the problem of creating such a characterization.

The following methods may be used to create a descriptive-storage arrangement which is optimal:

3.1-1 (i) Pass-the-buck (to the user).

(ii) Use a restrictive definition.

(iii) Enumerate all possibilities and select the one for which cost is least.

(iv) Derive an analytic result to indicate the best solution.

(v) Use a search procedure to determine a (nearly) optimal solution.

The first four techniques are traditionally employed. The last method has not been used extensively, although the following related results are known.

(i) Chu [CHU69] used linear programming to decide how to best allocate files in a special network.

(ii) Aspinall, Bell, and Rogers [ASPIN72] used the B&B method to decide where to allocate records in a direct-access file.

(iii) Ramamoorthy and Chandy [RAM70] used the B&B method to determine the best memory hierarchy for a multi-programmed computer system.

The first method of 3.1-1 involves having the users of an STDS system specify exactly which sets of descriptive storage are to be created. Except in limited situations, this technique is unsatisfactory because

(i) The users must be technically capable of deciding which sets to create; and

(ii) there may be conflicting opinions regarding which sets to create when more than one user is involved.

The second method of 3.1-1 is to use an STDS definition so restrictive that only one possibility for descriptive storage exists (and hence is optimal). For example, consider a Wong-and-Chiang descriptive storage where the additional restrictions are:

(i) The collection A of attributes is fixed; and

(ii) Sets are constructed using all the attributes in A.

In this case, there is only one collection of sets which satisfies this definition. The obvious drawback to this method is that another restrictive definition may be less expensive.

Another method is to enumerate the expected cost for all possible arrangements and then to select an arrangement for which cost is minimum. Again, except for limited cases, this technique is impractical because of the large number of choices. For example, an inverted file with n potential lists has 2^n alternatives.

The fourth method, when it applies, is the most desireable. An analytic result is the easiest method to determine which sets to create. Unfortunately, analytic results are known for only a few restrictive cases and so do not apply in general. An approach is as follows:

(i) Derive, by some method, a cost function for the system to be optimized.

(ii) Minimize this function.

For a particular system we shall show that it is theoretically possible to perform this operation, under the following assumptions:

(i) There is a finite collection of sets which may be defined; and

(ii) For every possible arrangement an expected

total cost, for a time interval, may be calculated. In this case, for n candidates for the defined sets, there are 2^n arrangements and we may define a cost function which is a pseudo-Boolean (pB) function. A pB function is a real-valued function with bivalent (0,1) variables. We may define the cost function as follows:

3.1-2 (i) Let S_i $(1 \leq i \leq n)$ be a bivalent variable with a value of one if the i-th candidate is defined and a value of zero otherwise. Denote by $\bar{S}_i$ the complementary value of S_i.

(ii) Then, there are 2^n terms, $\mathfrak{S}_1 \cdot \mathfrak{S}_2 \cdot \ldots \cdot \mathfrak{S}_n$, where $\mathfrak{S}_i$ $(1 \leq i \leq n)$ is S_i or $\bar{S}_i$. Arbitrarily arrange these terms in some order and denote the j-th term $(i \leq j \leq 2^n)$ by T_j.

(iii) Define a real number C_j $(1 \leq j \leq 2^n)$ as the expected cost of the arrangement in which the defined sets are exactly those sets which correspond to an uncomplemented symbol in T_j.

(iv) Then, the cost function is given by $\sum_{j=1}^{2^n} C_j \cdot T_j$.

This function specifies the cost of an arbitrary arrangement because:

(i) In the j-th arrangement $(1 \leq j \leq 2^n)$, let the variables S_i $(1 \leq i \leq n)$ have a value of one if the i-th variable is defined and zero otherwise. Then,

T_j evaluates as one and every other T_k ($1 \leq k \leq 2^n$, $j \neq k$) evaluates as zero.

(ii) Hence, $\sum_{k=1}^{2^n} C_k \cdot T_k = C_j$, for the j-th arrangement, which concludes the proof.

The substitution of $(1-S_i)$ for $\bar{S}_i$ ($1 \leq i \leq n$) results in a cost equation involving only operators, constants, and uncomplemented bivalent variables. Methods exist for the maximization of pB functions [HAMM72]. These techniques may be adapted to minimize the functions, or the cost functions, multiplied by negative one, may be maximized.

We note that the computation of C_j ($1 \leq j \leq 2^n$) is equivalent to enumerating all cases. Hence, where the enumerative method is unfeasible, the creation of a pB cost function is unfeasible, unless the function can be obtained in an alternative form in a reasonable number of steps.

The last method cited in 3.1-1 involves searching for the best descriptive-storage arrangement. A general formulation of the method is as follows:

(i) Create a collection of sets (called candidates) which may be defined; and

(ii) Eliminate (select) members of this collection which are not (are) in the optimal collection.

Since a finite number of cases exist, this method always works. To avoid considering all cases we must specify a method of determining when we have found the best collection. A systematic way of searching must be specified, which,

hopefully, will avoid enumerating all choices. Because of the generality of this method, we shall devote the following section to the consideration of it.

3.2 Search Procedures

As indicated in the preceding section, a general formulation of a search procedure is as follows:

3.2-1 (i) Create a collection of candidates; i.e., a collection of sets which may be potentially defined as descriptive storage.

(ii) Select (eliminate) members of this collection which are (are not) in the optimal collection.

Candidates are constructed from the queries which have entered the system. Thus we predict that the STDS system will be used in the future similarly to how it was used historically. Subsection 3.2.2 will be an investigation of how to construct the candidate collection.

The selection/elimination phase is an adaption of a well-known optimization procedure, the branch-and-bound method. Cost is the criterion by which candidates are accepted or rejected. Thus, 'small' sets are preferred to 'large' sets and 'frequently used' sets are preferred to 'infrequently used' sets. The following subsection will indicate how this optimization may be implemented.

3.2.1 Branch-and-Bound Adaptation

Within this subsection we shall assume that a candidate collection has been determined (the next subsection will discuss how to create such a collection). We shall assume that this candidate collection contains an optimal sub-collection; i.e., if the optimal sub-collection is used as descriptive storage, then the STDS system is expected to have a minimum total cost over an arbitrary period of time.

A survey of branch-and-bound implementations and applications, together with an extensive bibliography, is found in [LAWL69]. The branch-and-bound (B&B) method may be described as follows. Let $f(x_1,x_2,\ldots,x_n)$ be a function to be minimized where the i-th variable ($1 \le i \le n$) may assume an integral number (m_i) values. Thus, there is a set of $\prod_{i=1}^{n} m_i$ feasible valuations of the variables. The B&B method repeatedly partitions the set of feasible valuations into smaller and smaller subsets. There is associated, with every subset in the partition of feasible solutions, a lower bound for the minimum evaluation of f within that subset. As we perform each partitioning, we refine an upper bound for the minimum evaluation of f. Subsets for which the associated lower bound is greater than the upper bound for the minimum evaluation of f are clearly unfeasible and are excluded from further partitioning.

We shall adopt the B&B method to determine the sub-collection of candidates for which the expected cost (during an arbitrary time interval) of operating an STDS system is

minimal. We consider each of the n candidates to be a bivalent variable whose value is one when the candidate is defined in an STDS arrangement and whose value is zero otherwise. We denote the set of candidates by the collection $\{S_1, S_2, \ldots, S_n\}$.

We characterize each subset of valuations of the variables as a four-tuple <j,D,Cmin,X> where

2.2.1-1 (i) $D \subseteq \{S_1, S_2, \ldots, S_j\}$ is a collection of sets defined for all members of the subset. Sets with an index greater than j have not yet been considered with regard to this subset.

(ii) Cmin is a lower bound for expected cost of operating the STDS system when descriptive storage is composed of the set D and a subset of $\{S_{j+1}, S_{j+2}, \ldots, S_n\}$.

(iii) X is one of the symbols {u,p} where 'u' indicates that the subset is unfeasible and 'p' denotes that the subset is feasible or potentially contains the best sub-collection of candidates.

Thus, the subset <2,{S_1},436,'p'> is feasible and 436 is a lower bound for the expected cost of an STDS for which S_1 is defined and S_2 is not defined. We will continue to partition subsets of solutions which have the symbol 'p' and for which $j<n$. The partitioning process replaces a subset in which j

candidates have been considered with two subsets in which j+1 candidates have been considered. Our adaptation of the B&B algorithm is as follows.

3.2.1-2 (i) Start with one subset, <0,Ø,Z,'p'>, which represents all possible collections of candidates. Z is the manipulative cost when all candidates are defined. Set the initial value of a variable Cmax to the expected cost when descriptive storage consists of all candidates.

(ii) Select a subset <j,D,Cmin,'p'> for which j<n. If no such subset exists, then the optimal collection of candidates is D, where there exists a subset <n,D,Cmin,'p'>.

(iii) Replace the selected subset by the two subsets:

<j+1,D,Cmin,'p'>

<j+1,D∪{S_{j+1}},Cmin,'p'>

(iv) Replace the value of Cmax by the minimum of the present value of Cmax and the expected cost where descriptive storage is the collection D ∪ {S_{j+1}}.

(v) For any subset, <j,D,Cmin,'p'>, if Cmin>Cmax, then change the symbol 'p' to 'u'.

(vi) Repeat, starting at step (ii).

We shall neglect the cost of storing the file in all total cost computations. If this cost is to be included, then the constant value of this cost may be added to the total cost computations specified above.

Recalling (3.2.1-1) the definition of Cmin, we calculate Cmin, in the subset $\langle j,D,Cmin,x\rangle$ as the sum of

3.2.1-3 (i) the expected cost to store the members of D;

and (ii) the expected cost of retrieval when descriptive storage is defined to be $D \cup \{S_{j+1},S_{j+2},\ldots,S_n\}$.

We assume that increasing descriptive storage will not decrease the manipulative (retrieval) cost. We have argued that this assumption is reasonable in section 1.3.

Cmax always represents the expected cost of the STDS when descriptive storage is a subset of the candidate collection. Thus, Cmax represents an upper bound for the minimum cost of operating the STDS system. Moreover, when the optimization terminates, the value of Cmax is this minimum expected cost.

In order to estimate manipulative (retrieval) cost for a given storage collection, we must formulate how the STDS system will be used during the time interval in question. One formulation is in terms of two sets F and Q such that

3.2.1-4 (i) $Q = \{q_1,q_2,\ldots,q_k\}$ is a collection of queries for which the system is expected to find retrieval sets during the time interval in question; and

(ii) $F = \{f_1, f_2, \ldots, f_k\}$ is a set of real numbers such that f_i $(1 \leq i \leq k)$ is the frequency that q_i is expected to be posed during the time period.

Suppose that $R(q_i, X)$ is the expected cost of finding a retrieval set for the i-th query $(1 \leq i \leq k)$ when $X \subseteq \{S_1, S_2, \ldots, S_n\}$ is used as descriptive storage. Then, the expected manipulative cost for the time period is given by

3.2.1-5 $$M(X) = \sum_{i=1}^{k} f_i \cdot R(q_i, X)$$

Let $l(S_j)$ denote the cost of storing the set S_j $(1 \leq j \leq n)$ during the time period in question. Then, for a subset $\langle j, D, Cmin, x \rangle$ of feasible collections, we compute a lower bound on minimum cost as

3.2.1-6 $$Cmin = \sum_{S \in D} l(S) + M(D')$$

$$= \sum_{S \in D} l(S) + \sum_{i=1}^{k} f_i \cdot R(q_i, D')$$

where $D' = D \cup \{S_{j+1}, S_{j+2}, \ldots, S_n\}$

Recalling the general formulation (3.2.1-2) of the B&B algorithm, we note that worst computational case occurs when no subsets can be marked unfeasible (step v) during the algorithm. In this situation, steps (ii) to (vi) are performed 2^{n-1} times. The computations for expected costs involve k queries and so are of this order. Hence, an upper bound on the number of computations is of the order $k \cdot 2^n$. The actual number of steps can be reduced by:

(i) using "good" branching rules; i.e., the rules by which the next subset of feasible collects is chosen to partition.

(ii) considering the candidates in a "good" order

(iii) allowing, as optimal, subcollections of candidates for which the expected cost differs from the minimum by a specified tolerance.

(iv) performing a number of "small" optimizations instead of one "large" one.

Before we investigate these techniques, we shall consider an example.

We summarize ficticious costs for four queries and three candidates in Table 3.2.1-7. Each row of the table consists of a binary sequence followed by six costs. The binary numbers represent whether or not the corresponding candidates are defined. The first four costs are the expected costs to perform retrievals for the four queries when descriptive storage is composed of the sets for which the corresponding binary number is one. The last two columns give expected manipulation and storage costs for the corresponding descriptive storage. Thus, for the row headed by the binary sequence 100, the expected cost to retrieve the first query is 500 and the expected manipulation cost is 780 (780 = (4x50) + (3x100) + (4x40) + (2x100)). The costs to store the three candidates are respectively 100, 200, and 100. The expected frequencies which the four queries will entered during the time interval in question are respectively 4, 3, 2, and 2.

Table 3.2.1-7 Summary of Costs for example 3.2.1-8

Descriptive Storage Sets			Query Costs				Expected Manip. Cost	Expect Storage Cost
1	2	3	1	2	3	4		
0	0	0	100	100	100	100	1100	0
0	0	1	100	20	50	60	680	100
0	1	0	100	100	100	600	1020	200
0	1	1	100	20	50	30	620	300
1	0	0	50	100	40	100	780	100
1	0	1	20	20	10	60	280	200
1	1	0	50	100	40	60	700	300
1	1	1	20	20	10	30	220	400

Cost to store sets: 100, 200, 100

Query frequencies: 4, 3, 2, 2

Example 3.2.1-8 is a B&B optimization based upon the data in table 3.2.1-7. The computations are summarized by the tree in Diagram 3.2.1-9. The nodes in the tree are the subsets. There are two branches for each iteration of the algorithm. Branches emanate from a node to the two nodes created by partitioning the original node. For legibility, we have omitted the symbols 'p' and 'u' when denoting subsets in the diagram.

In example 3.2.1-8, more than one feasible subset could have been partitioned in iterations 2 and 3. We arbitrarily partitioned the subset which had the minimum lower bound on cost. Branching rules are used to decide which subset to partition when more than one such subset exists. Example of branching rules include:

3.2.1-10 (i) Partition feasible subsets according to the order in which they were created (FIFO rule).

(ii) Partition feasible subsets in the opposite order to which they were created (LIFO rule).

(iii) Partition feasible subsets according to some prediction that an optimal collection occurs in a subset (heuristic approach).

An example of (iii) is the rule used in example 3.2.1-8. When the FIFO and LIFO rules were applied to the costs in table 3.2.1-7 five and four iterations were required, respectively. A summary of the computations for these two rules is found in diagrams 3.2.1-11 and 3.2.1-12, respectively. The effect of

EXAMPLE 3.2.1-8: Sample B&B Optimization

INITIALLY

SUBSET (0) <0,Ø,220,'p'>

Cmax = 1100

ITERATION (1)

- partition subset (0) into subsets (1) and (2):
  - (1) <1,Ø,620,'p'>
  - (2) <1,{S,},320,'p'>

ITERATION (2)

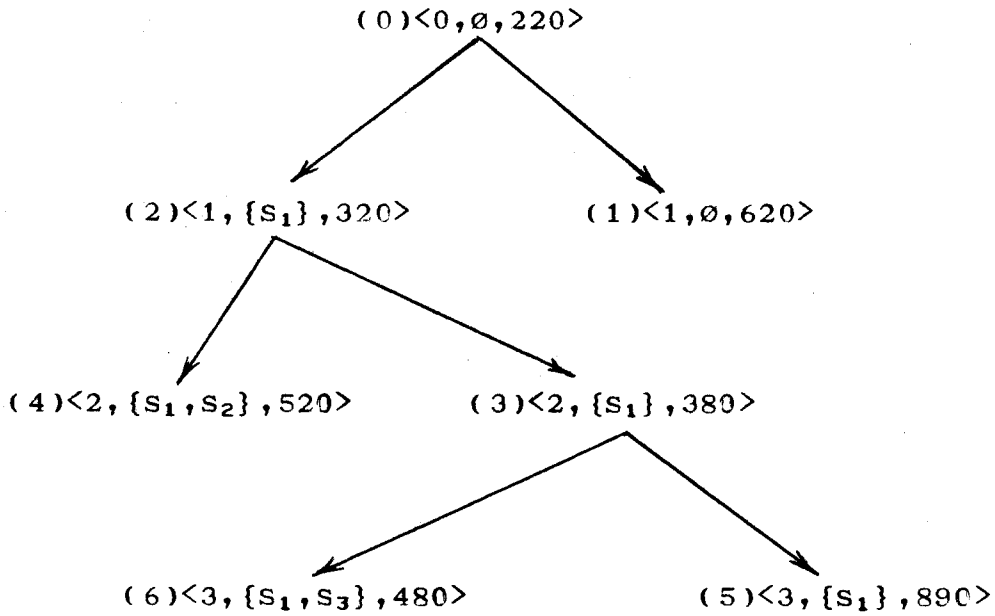
- partition subset (2) into subsets (3) and (4):
  - (3) <2,{S,},380,'p'>
  - (4) <2,{S_1,S_2},520,'p'>
- Cmax = 880

ITERATION (3)

- partition subset (3) into subsets (5) and (6)
  - (5) <3,{S,},890,'p'>
  - (6) <3,{S_1,S_3},480,'p'>
- Cmax = 480
- subsets (1), (4), (5) are marked unfeasible.

The algorithm terminates because only subset (6) is feasible and it cannot be partitioned further. The optimal descriptive storage consists of the first and third sets. The expected cost of this arrangement is 480.

Diagram 3.2.1-9 Tree summarizing calculations for example 3.2.1-8



Optimal collection is subset(6): descriptive storage is composed of sets $\{S_1, S_2\}$ and has an expected cost of 480.

Diagram 3.2.1-11: Tree summarizing calculations for example 3.2.1-8 when FIFO rule is used.

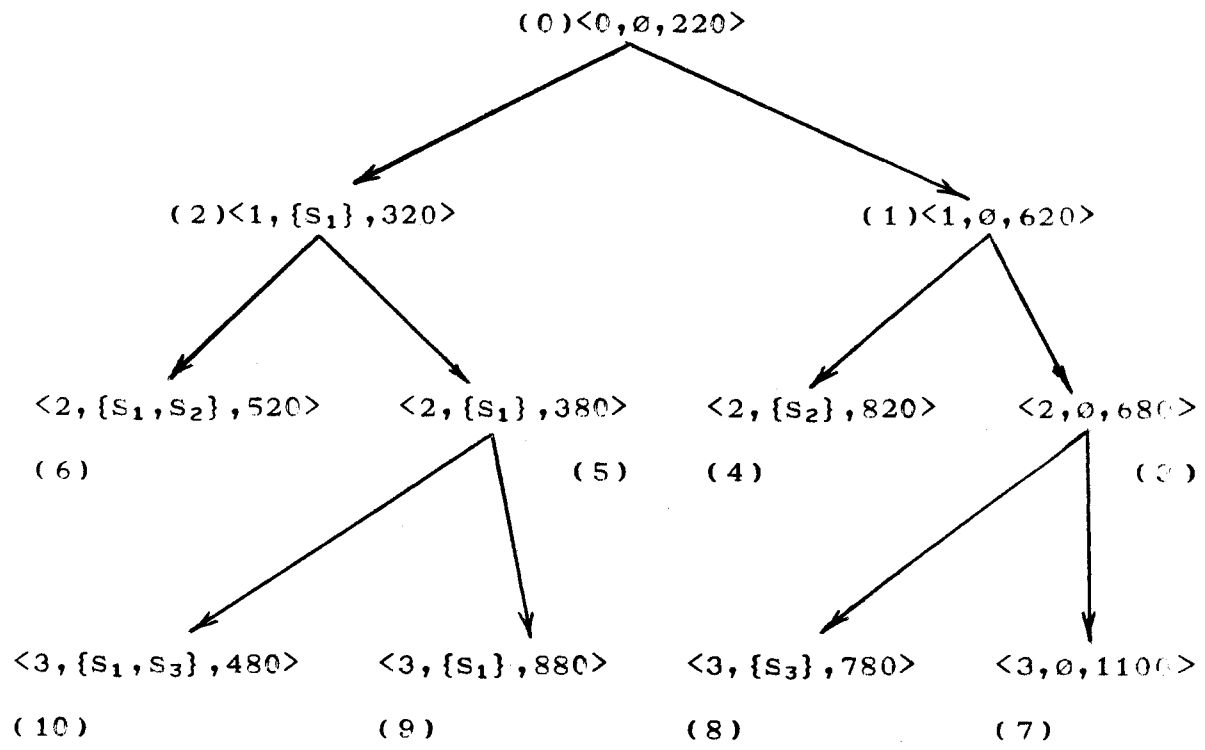
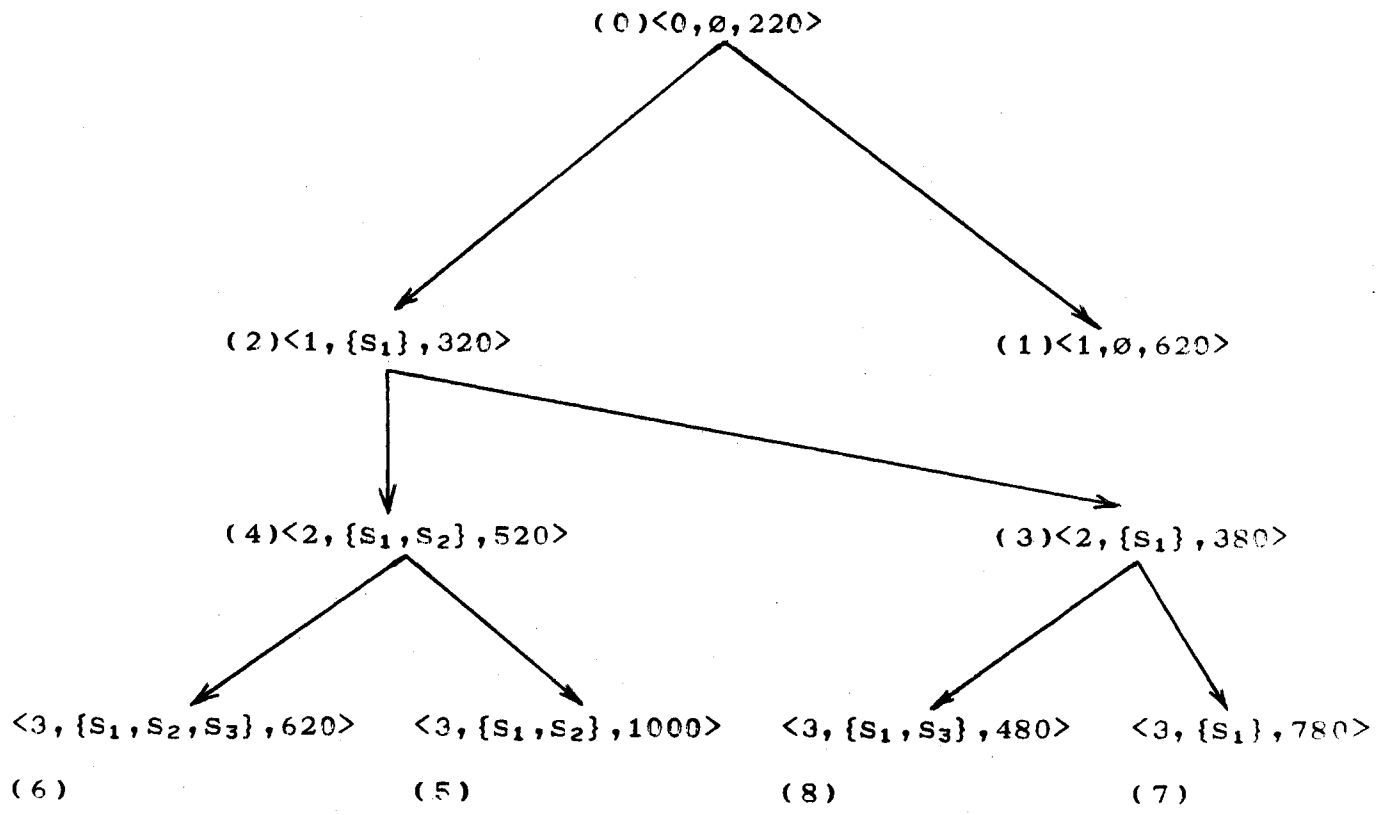


Diagram 3.2.1-12 Tree summarizing calculations for example 3.2.1-8 when LIFO rule is used.



the FIFO rule is to favour the partitioning of larger subsets. Conversely, the LIFO rule tends to favour the partitioning of smaller subsets. Either of these two rules may be inefficient because the partitioning of another set may cause the value of Cmax to decrease so that the original subsets would be marked unfeasible. The heuristic approach, outlined in 3.2.1-10, appears to be the most promising. The criterion applied in example 3.2.1-8 (choosing the feasible subset with the least lower bound for cost) favours partitioning larger subsets, although to a lesser degree than the FIFO rule. It is anticipated that experience with a number of particular systems will indicate a number of more efficient rules.

The number of iterations during B&B optimization is affected by the order in which candidates are considered. For example 3.2.1-8, if the sets were considered in the order S_3, S_1, S_2, then four iterations would be required to determine the optimal collection $\{S_2, S_1\}$. It is desireable that the sets be considered in an order whereby large subsets of collections are marked unfeasible, thereby avoiding useless computations and partitioning. Suppose, for example, that a candidate has a relatively high storage cost and that there is a relatively small difference in manipulation cost, regardless of whether or not the candidate is defined. Then, the subsets of collections in which the candidate is defined are likely to have a higher value of Cmin than the subsets where the candidate is not defined. The opposite situation occurs when a candidate has a relatively high difference between

manipulation costs and a relatively low storage cost. Consider, for example, the costs depicted in table 3.2.1-7. We summarize the effects of defining a candidate in table 3.2.1-13. For each of these candidates we present the average cost when candidates are defined and not defined. The differences in these averages is compared with the storage costs for the candidates.

TABLE 3.2.1-13 Average Cost Summary

			Candidates		
	Manipulation		(1)	(2)	(3)
(i)	Costs:	Not Defined:	1005	810	1050
(ii)		Defined:	645	740	600
(iii)	Difference in manipulation costs:		360	70	450
(iv)	Storage costs		100	200	100
(v)	Absolute difference between (iii) and (iv)		260	130	350

Comparing the absolute differences in table 3.2.1-13, we conclude that the best order to consider the sets is S_3, S_1, S_2.

Of course, in an optimization with many candidates it is generally impractical to calculate the expected costs and derive a table similar to 3.2.1-13. The analysis, however, indicates several situations in which "good" orders may be found. When relatively few of the candidates are to be included in the optimal collection, a "good" order is to consider the candidates in ascending order of their storage costs. Similarily, when relatively many candidates are to be

included, the opposite order is preferred.

As indicated in [LAWL69], a third method to reduce the amount of computation is to accept "nearly optimal" solutions. In other words, we agree to accept a sub-collection of candidates for which the expected cost differs from the minimum expected cost by no more than a prescribed amount. Suppose, for example, that a percentage of error e is acceptable and that a sub-collection of candidates has expected cost Cmax. Then, any subsets of collections, which has a corresponding value of Cmin satisfying $(1+e)\cdot Cmin > Cmax$, can be excluded from further partitioning. If all subsets may be eliminated in this manner, then the "nearly optimal" solution is a sub-collection of the candidates for which the expected cost is Cmax. Another way of proceeding is to use an "adaptive" computational scheme in which the amount of acceptable error increases during the optimization. For example, we could attempt to find nearly optimal collections during time intervals T/2, T/4, T/8,... using acceptable errors of 0, e, 2e, ... In this manner, a solution is always determined in a time interval of length T. We have not investigated this third method, as we have been able to perform all optimizations using a fourth technique, partitioning.

We have developed the idea of partitioning in order to do a number of small optimizations instead of a single large one. As we shall indicate, this approach substantially reduces the maximum number of calculations in the B&B algorithm. Our

computational experience indicates that the actual number of steps is also reduced. The problem is to decide how to partition the candidates, queries, and frequencies so that the union of the sub-optimal solutions is also the optimal solution when the candidates, queries, and frequencies are considered jointly. Before we develop the partitioning method and informally justify it, we shall illustrate the computational advantages of it.

Suppose that n candidates may be partitioned into p groups where the i-th group ($1 \le i \le p$) has n_i members. Then, performing B&B optimizations for each of the p partitions, the maximum number of iterations is

3.2.1-14 $$\sum_{i=1}^{p} k \cdot 2^{n_i} = k \cdot \sum_{i=1}^{p} 2^{n_i} \le k \cdot 2^{n}$$

where there are k queries characterizing system usage

and $n = \sum_{i=1}^{p} n_i$.

If the k queries could also be partitioned so that the i-th partition ($1 \le i \le p$) of candidates was characterized by k queries, then the maximum number of steps would be

3.2.1-15 $$\sum_{i=1}^{p} k_i \cdot 2^{n_i} \le k \cdot \sum_{i=1}^{p} 2^{n_i}$$

where $n = \sum_{i=1}^{p} n_i$ and $k = \sum_{i=1}^{p} k_i$.

For example, if 10 candidates were partitioned into groups of 2, 5, and 3 and if the 8 queries were partitioned into groups of 4, 2, and 2 queries, then,

3.2.1-16 $96 = (4 \bullet 2^2) + (2 \bullet 2^5) + (2 \bullet 2^3)$

would be the maximum number of iterations, compared to

3.2.1-17 $8,192 = 8 \bullet 2^{10}$

iterations without partitioning.

To decide how to partition the candidates and queries we introduce the concept of dependence between candidates. Intuitively, two or more candidates are dependent upon one another if the inclusion or exclusion of one candidate in descriptive storage changes how the other candidates are used. For example, suppose the retrieval set for a query can be determined using the collections $\{S_1, S_2, S_3\}$ or $\{S_4, S_5\}$ where the former is used in preference to the latter. Then, the exclusion of set S_1 from descriptive storage would mean that whenever the query in question is entered, the latter collection would be used (if sets S_4, S_5 are defined) to determine the retrieval set.

In general, each query q in the set Q may be associated with a sub-collection of the candidate collection. This sub-collection consists of all candidates which may be used to determine the retrieval set for q. We denote this sub-collection as S(q). For convenience, we define a relation θ as follows.

3.2.1-18 Definition For queries $q_1, q_n \in Q$, $q_1 \ \theta \ q_n$ if and only if there exists a finite sequence of queries $q_1, q_2, \ldots, q_n$ such that

(i) $q_i \in Q$, for $1 \leq i \leq n$

(ii) $S(q_i) \cap S(q_{i+1}) \neq \emptyset$, for $1 \leq i < n$

As is easily proven, θ is symmetric, reflexive, and transitive, and so is an equivalence relation. Hence, θ may be used to partition Q into p equivalence classes. Furthermore, as is easily demonstrated, θ partitions the set of candidates into p groups as follows. Let the i-th partition of Q be $\{q_1^i, q_2^i, \ldots, q_{k_i}^i\}$. Then, the i-th partition of $\{S_1^i, S_2^i, \ldots, S_n^i\}$ consists of all candidates $S(q_1^i) \cup S(q_2^i) \cup \ldots \cup S(q_{k_i}^i)$. We will now prove that these p groups of candidates are disjoint.

3.2.1-19 Proof

Assume a candidate S occurs in the i-th and j-th group. Then, there exist queries q_1 and q_2 in the i-th and j-th partitions, respectively, such that $S \in S(q_1)$ and $S \in S(q_2)$. Hence, $S \in S(q_1) \cap S(q_2)$ and so $q_1 \; \theta \; q_2$. Thus, q_1 and q_2 are in the same equivalence class and so i=j. This completes the proof.

We have not yet considered candidates which occur in none of the sub-collections associated with the queries of Q. For completeness, we could include them in any of the groups. It is preferable, however, to eliminate them immediately from the candidate collection: none of these candidates will occur in an optimal sub-collection since they are never expected to be

used for retrieval and will only increase the storage cost. Similarly, if a query exists such that none of the candidates can be used to create its retrieval set, then that query can be eliminated: it will always be satisfied by sequentially processing the file.

We will demonstrate that the method of partitioning, outlined above, determines a sub-collection of the candidates for which the expected cost is minimum. Let there be p partitions where the i-th partition ($1 \le i \le p$) has a candidate sub-collection $\mathcal{S}_i$ and a set of queries Q_i. Denote by f(q) the expected frequency of an arbitrary query q in the time interval in question. For any candidate subcollection X, let P(q,X) be the expected cost of performing a retrieval for a query q. We assume that the addition to descriptive storage of sets which do not occur in S(q) will not change the value of R(q,X). Thus, if a collection of candidates Y satisfies $Y \cap S(q) = \emptyset$ then, R(q,X) = R(q,X∪Y). Now for the i-th partition ($1 \le i \le p$), let Z_i be a sub-collection of $\mathcal{S}_i$ such that

3.2.1-20 $$\sum_{s \in Z_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,Z_i)$$

is minimum. Let X be any sub-collection of the set of candidates and define $X_i = X \cap \mathcal{S}_i$ for $1 \le i \le p$. Then, by the definitions of X_i and Z_i, we have ($1 \le i \le p$)

3.2.1-21 $$\sum_{s \in X_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,X_i) \ge \sum_{s \in Z_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,Z_i)$$

But, for $q \in Q_i$, $S(q) \cap X_i \neq \emptyset$ if and only if i=j. Hence, for $q \in Q_i$, $R(q,X_i) = R(q,X)$. Similarily, for $q \in Q_i$, $R(q,Z_i) = R(q,Z)$

where $Z = \bigcup_{i=1}^{p} Z_i$. Hence, from 3.2.1-21 we derive ($1 \le i \le p$)

3.2.1-22 $$\sum_{S \in X_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,X) \ge \sum_{S \in Z_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,Z)$$

and summing 3.2.1-22 over the p partitions, we get

3.2.1-23 $$\sum_{i=1}^{p} \left\{ \sum_{S \in X_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,X) \right\} \ge \sum_{i=1}^{p} \left\{ \sum_{S \in Z_i} l(S) + \sum_{q \in Q_i} f(q) \cdot R(q,Z) \right\}$$

or

3.2.1-24 $$\sum_{S \in X} l(S) + \sum_{q \in Q} f(q) \cdot R(q,X) \ge \sum_{S \in Z} l(S) + \sum_{q \in Q} f(q) \cdot R(q,Z)$$

This proves that the sub-collection Z is an arrangement of descriptive storage for which the expected cost is minimum. Furthermore, the sum of the expected costs (3.2.1-20) over the p partitions is this minimum expected cost. Thus, the partitioning method determines a descriptive storage with the same expected cost as the one found without using partitioning. Computational experience (see Appendix D) suggests that the partitioning method finds an optimal solution in fewer steps then the non-partitioned case. Of course, it is recommended that techniques, such as "good" branching techniques, be employed in the optimization of each partition.

We have assumed throughout this sub-section that the candidate collection has been determined. The following sub-section shall indicate methods to create this collection and the characterization of expected system usage.

3.2.2 Creation of Candidates

This sub-section is concerned with how to create the collection of candidates and how to create the characterization of system usage. To serve as a prediction of how the system will be used, we create two sets:

(i) the set Q of the queries which have been entered the system;

and (ii) the set F of the frequencies which the queries in Q are expected to be posed during the time interval of the optimization.

The candidate collection can then be created using these two sets.

This choice for the collection of queries to characterize system usage could provide a set of queries which is too large to be effectively used in the B&B optimization. Hence, we propose three methods to reduce this number to a workable size:

3.2.2-1 (i) Random Selection

(ii) Probabilistic Selection

(iii) Classification

By the term 'workable size' we mean that the maximum or expected number of steps in the proposed B&B optimization be reasonable.

Recalling the Feedback Model (section 2.5), we proposed that the retrieval process be augmented with a measurement

facility. This facility would record the queries which entered the system, in addition to the usual accounting information. To each unique query we may assign a probability in a number of ways. The simplest method to estimate the probability for a query is to take the quotient of

3.2.2-2 (i) the frequency which a query was entered;
and (ii) the sum of (i) for all queries.

This method ignores when the query was entered. If the usage pattern of the system is changing, it may be preferable to bias these probabilities so that queries which were entered recently would tend to have larger probabilities. Thus, weighted frequencies could be used where the time of entry determines the weight added to the frequency. For example, the weights could be aged exponentially, where the weight for a query entered n days ago is W^n ($0 \leq W \leq 1$). Thus, the weighted frequency for a query which was entered twice, 4 and 8 days ago, would be ($W^4 + W^8$). The weight W to use, and the intervals of time to age the frequencies, we shall leave unspecified. These values would be parameters of the particular system in which they are used.

As indicated previously, the number of queries can be reduced in at least three ways. The simplest is random selection. Using a device such as a random-number generator, k queries are selected to consititute the collection. The frequencies corresponding to this set are then used to create the corresponding probabilities.

A second method is probabilistic selection. In this situation, the k queries with the highest frequencies are selected. This method is preferred to random selection when the range of frequencies is large. When all frequencies are of about the same magnitude, random selection is likely to be as effective as probabilistic selection. Again, as with random selection, the frequencies of the k queries are used to estimate the corresponding probabilities.

The probabilities which we have calculated for each query are estimates of the probability that an arbitrary query which enters the system is the query in question. An estimate of the frequency that a query will be entered during a time interval is the product of the above estimated probability and the number of queries expected to be entered during the time interval in question.

Before we consider the third method (classification of queries), we shall investigate how to create candidates for descriptive storage. We create candidates based upon the description of how the system will be used. One choice for the candidate collection is the collection of sets which corresponds to the retrieval sets for the queries selected in the description. Consider, however, two such sets S_1 and S_2 which are the retrieval sets for two queries. Less storage space (and less storage cost) might be required to store the three sets:

<u>3.2.2-3</u> $S_3 = \{d | d \in S_1 \text{ and } d \in S_2\}$

$S_4 = \{d | d \in S_1 \text{ and } d \notin S_2\}$

$$S_5 = \{d \mid d \notin S_1 \text{ and } d \in S_2\}$$

In general, there is a tradeoff between the reduction of cost due to reduced storage requirements and the additional cost of combining these "small" sets during retrieval.

It is well-known fact [WOOD68] of Boolean algebra that there exist 2^{2^k} distinct Boolean expressions of k bivalent variables. Considering each of the k queries as a bivalent variable, there exists a maximum of 2^{2^k} potential members of the candidate collection. This is clearly an unreasonable number except in the unusual situation where there is a very small number of queries.

In some cases, added restrictions on the type of sets allowed may reduce the number of potential candidates to a reasonable number. For example, if the STDS system was restricted to inverted lists whose definition involved only a single attribute, then this number may already be reasonable. In other cases, we propose that restrictions be added to reduce the number of possible candidates.

One manner of restricting the possible candidates is to limit the definitions of the sets which are used as candidates. For example, the definition of the sets may be limited to a specific format, as is the case with inverted files. Another possibility is to allow only atoms, in the terms of Wong and Chiang [WONG71], to be candidates. The queries may also be used to create a restricted candidate

collection. For example, we could allow non-null sets which correspond to the retrieval sets for the queries and the pair-wise intersection of them (a maximum of $k \cdot (k+1)/2$ candidates).

Another method of restriction is to allow only candidates which are judged to be sufficiently useful. Intuitively, larger sets are less useful than smaller sets. Also, a set which is used frequently for retrieval puposes is more useful than one which is accessed infrequently. Thus, we could arbitrarily exclude all candidates whose size exceeded some boundary value. Similarly, by estimating the probability of accessing a set during an arbitrary retrieval, we could include as candidates only sets whose probability was greater than some arbitrary value. In general, this preliminary screening of candidates ignores any dependencies between sets. As such, the estimates of the probabilities, for example, would be very crude.

We have indicated these restrictions to demonstrate that there are reasonable ways to proceed when a general STDS system is implemented. In the particular systems, referenced in Chapter Four, such procedures are generally unnecessary; the definition of these systems usually restricts the candidates to a reasonable number.

Whatever the manner used to create the collection of candidates and the description of system usage, the optimization procedure must be able to determine an optimal collection in a reasonable number of steps. We have shown that the maximum number of steps is of the order $k \cdot 2^n$, where

there are k queries and n candidates. Experience may indicate to a system administrator that the actual number of steps is of a different magnitude, for a particular system. In any case, the system administrator is likely to have an estimate of the number of steps which may be feasibly performed. Hence, it is possible to compute reasonable values of k and n from this number of steps, or these values may be directly passed to the candidate-creation phase of the optimization procedure.

As indicated in the previous subsection, substantial reduction in the number of actual steps performed during an optimization is possible when partitioning is used. Unfortunately, in many cases only a few (possibly only one) partitions exist. For example, suppose that k queries characterize system usage and that we create a candidate collection as follows. Let S_1 and S_2 be the retrieval sets for two arbitrary queries. Then, we include in the candidate collection all non-null sets defined as

3.2.2-4 $\{d \mid d \in S_1 \text{ and } d \in S_2\}$

$\{d \mid d \in S_1 \text{ and } d \notin S_2\}$

$\{d \mid d \notin S_1 \text{ and } d \in S_2\}$

Thus, if all sets 3.2.2-4 are non-null, we have $3k(k+1)/2$ candidates and partitioning of the queries may yield only one partition.

Thus, it may be desireable to create the query and candidate collections in a manner in which partitioning will

be effective. An approach is to select p groups of queries and then to construct candidates for each of these groups in a manner that will determine p partitions of the candidates. It is natural to require that the queries within a group be, in some sense, similar. Hence, the application of classification theory appears appropriate.

Classification methods have been used in toxonomy [WALL68], medical research [BONN64], document retrieval systems [SALT63, SALT69], and are also of theoretical interest [LANCE66, LANCE67A, LANCE67B, JARD68, LANCE68, SHEP68, JARD71, VANR71, LING72, SIB73]. Alternative terms for 'classification methods' include 'clustering', 'cluster analysis', 'toxometric analysis', 'similarity analysis', and 'association analysis'. The problem is formulated as follows: given a set of k objects with pair-wise similarity/dissimilarity measures, find groups or clusters of objects which are similar. The techniques are conveniently divided into hierarchical and non-hierarchical methods.

Hierarchical methods are either divisive or agglomerative. Agglomerative methods may be summarized by the following algorithm [CUNN72]:

3.2.2-5 (i) Search for the two most similar objects (clusters), say p and q.

(ii) Fuse the objects in p and q to form a new cluster, say n.

(iii) To every other object or cluster l, recompute the similarily/dissimilarity measure between n and l.

(iv) Repeat steps (i) to (iii) until only one cluster remains.

This procedure generates k partitions in a hierarchy where there are k objects to be grouped. Divisive methods are similar except that all the objects are initially put into one group and each step of the algorithm involves splitting a cluster into two clusters. A study [CUNN72] has compared various agglomerative hierarchical methods.

Hierarchical methods find the most efficient step at each stage of the algorithm and so may be thought of as optimizing the route. As has been noted [LANCE66], this procedure sacrifices some homogeneity of the groups. Non-hierarchical methods, on the other hand, attempt to maximize the similarity of items within a group. These techniques are characterized by four processes [LANCE67B].

3.2.2-6 (i) a method of initiating clusters

(ii) a method of allocating new elements to existing clusters and/or fusing existing clusters

(iii) a method of determining when further allocation is unprofitable

(iv) a method of reallocating some or all the elements to existing clusters when the main classification process is completed.

It is noted [LANCE67B] that all systems involve (i) and (ii) but that (iii) and/or (iv) may be lacking in a particular system. Specific strategies are outlined in [LANCE67B, JARD68, LING72, VANR71, SIB73].

It is beyond the scope of this research to evaluate the various strategies. It is sufficient to indicate that a number of methods exist and may be adapted to the problem of grouping queries, provided a similarity/dissimilarity measure can be calculated for any two queries. We will propose such a measure and illustrate its use in an example.

For any two queries q_1 and q_2, the similarity/dissimilarity coefficient $d(q_1, q_2)$ should satisfy the following conditions:

3.2.2-7 (i) $d(q_1,q_2) = d(q_2,q_1)$

(ii) $d(q_1,q_1) = 1$

(iii) $1 \geq d(q_1,q_2) \geq 0$

Before definition of the coefficient, we shall discuss the notion of similarity between queries. In the STDS $\langle I,A,D,S \rangle$, the retrieval set R(q) for a query consists of exactly those data items $d \in I$ for which the query expression is true, i.e.,

3.2.2-8 $P(q) = \{d \mid (d \in I)(E(d,q)=1)\}$

We consider the similarity between two queries to be determined by the degree to which their retrieval sets correspond. As such, a crude measure might be the probability that an arbitrary data item occurs in the intersection of the retrieval sets. Although this measure satisfies the criteria 3.2.2-7, it suffers from the disadvantage that the value increases as the probabilities of an arbitrary data item occuring in the retrieval sets increase. This effect is alleviated if we use the measure

3.2.2-9 $d(q_1,q_2) = P(q_1 \wedge q_2)/P(q_1 \vee q_2)$

where $P(q)$ is the probability that an arbitrary data item is true for the Boolean expression q. Thus, we have proposed, as a measure of dissimilarity, the conditional probability that an arbitrary data item satisfies both queries, given that it satisfies one of them. These probabilities may be obtained by direct examination of the data items or by estimation.

To exemplify the classification method we present the following example. We will classify five queries into two groups. The queries are Boolean expressions where attributes are represented by small letters.

3.2.2-11 Queries to be Classified

$q_1 = a \vee b$

$q_2 = a \vee c$

$q_3 = c \vee d$

$q_4 = d \vee e$

$q_5 = c \vee d \vee e$

We shall assume that the probabilities of an arbitrary data

item having an arbitrary attribute are known:

3.2.2-12 **Probabilities of Data Item Having Attribute**

a : .3

b : .25

c : .1

d : .2

e : .15

We assume that the attributes are uniformly distributed among the data items and are distributed independently of one another. This is sufficient information to compute dissimilarity measures, outlined above. For example, we have

$$3.2.2\text{-}13 \quad d(q_1,q_2) = P(q_1 \wedge q_2)/P(q_1 \vee q_2)$$
$$= \{P(q_1)+P(q_2)-P(q_1 \vee q_2)\}/P(q_1 \vee q_2)$$
$$= \{.475+.37-.5275\}/.5275$$
$$= .6019$$

where we have calculated, for example

$$3.2.2\text{-}14 \quad P(q_1 \vee q_2) = P(a \vee b \vee c)$$
$$= 1 - P(\bar{a} \wedge \bar{b} \wedge \bar{c})$$
$$= 1 - P(\bar{a}) \cdot P(\bar{b}) \cdot P(\bar{c})$$
$$= 1 - (.7) \cdot (.75) \cdot (.9)$$
$$= .5275$$

Table 3.2.2-15 gives all the pairwise dissimilarity measures.

3.2.2-15 Pairwise dissimilarity measures for example

QUERIES	(2)	(3)	(4)	(5)
(1)	.60190	.21383	.23639	.27155
(2)		.31048	.20714	.32619
(3)			.54639	.72165
(4)				.82474

We shall use the hierarchical clustering method (3.2.2-5) where, when we fuse groups i and j into a group k, the dissimilarity measure d(k,l) between groups k and l is defined as

3.2.2-16 $d(k,l) = \{d(i,l)+d(j,l)\}/2$

Initially we have five groups of one query each. According to the measure, groups 4 and 5 are most similar and so are fused. After recomputing the appropriate measures, we have pairwise dissimilarity measures as follows.

3.2.2-17 Dissimilarity measures after first iteration

GROUPS	(2)	(3)	(4,5)
(1)	.60190	.21383	.25397
(2)		.31048	.26662
(3)			.63402

Continuing, we fuse group (3) with the group (4,5) and the dissimilarity measures become:

3.2.2-18 Dissimilarity measures after second iteration

GROUP	(2)	(3,4,5)
(1)	.60190	.23390
(2)		.28855

On the last step, we fuse groups (1) and (2) and we are left with two groups {1,2} and {3,4,5}.

At the point, candidates could be created for the two groups and then two B&B optimizations could be performed. Each optimization determines an optimal sub-collection of the candidates, for the partition in question. According to the argument in 3.2.1, the union of these optimal sub-collections is the descriptive storage for which total cost is expected to be minimum, provided no candidate in one partition may be used to determine the retrieval set for a query in another partition. If a candidate may be used in more than one partition, this result is not necessarily true. For example, consider the partitioning of queries in the previous example. A candidate whose definition is $a \vee b \vee c \vee d \vee e$ would rejected in favour of a candidate with definition $a \vee b \vee c$ in the first partition. Similarily, a set with defintion $c \vee d \vee e$ would be preferred in the second partition. A rejected candidate, however, may be the best choice when the five queries are considered jointly.

Of course, it is possible to choose a descriptive storage, given a partition of the queries, as follows:

3.2.2-19 (i) Create the candidate collection for each partition, disregarding queries in other partitions.

(ii) Select the optimal sub-collection of candidates in each partition.

(iii) Choose descriptive storage to be the union of these optimal sub-collections.

In fact, this approach is to be preferred since the alternative is to omit candidates which apply to more than one partition. When these candidates are excluded, the total expected cost may be greater than if they were included. As was shown in section 3.2.1, when these candidates are not included, total expected cost equals the sum of the expected costs for the partitions. The inclusion of these candidates may decrease the cost for some or all of the partitions. The sum of these costs is an upper bound for total expected cost because:

(i) Total expected storage cost is less than or equal to the sum of the expected storage costs in each partition, if a candidate is selected in more than one partition.

(ii) The expected cost to create the retrieval set for any query does not increase when candidates from other partitions may be used.

Thus, the classification method has the computational advantages of partitioning. It may, however, determine a

descriptive storage for which the expected cost is not minimum. Hence, this method should be employed only when a very large number of queries and/or candidates is to be used in a B&B optimization.

In summary, this subsection has been concerned with the creation of

(i) an estimate of how the STDS system is used for retrieval; and

(ii) a collection of candidates for descriptive storage, based upon (i).

Our approach has been to outline a number of techniques and, where applicable, to indicate when the alternative methods are preferable. This general discussion has been intended to demonstrate that the Branch-and-Bound approach is feasible in a wide range of STDS situations.

3.3 SUMMARY

This chapter has been concerned with a general method to select descriptive storage for an STDS system. It is anticipated that many of the particular techniques may be applied to file systems or data bases which do not strictly adhere to the STDS definition.

We have developed and proposed techniques whereby the branch-and-bound algorithm may be adapted to select the optimal (nearly-optimal) sub-collection of sets, given a collection of candidates. The problem of how to choose a candidate collection was also discussed. In some cases we would determine a nearly-optimal solution because the calculation of the minumum solution would involve too many steps. Essentially, this is a trade-off of the cost of optimization against the saving in system cost.

These developments are important because of their generality. Previously no general method has been proposed.

Chapter 4: Particular STDS systems

4.1 Introduction

This chapter shall be concerned with the application of the theory in chapters two and three to particular file systems. In summary, our results will be applied in two ways: the generalization of the retrieval mechanism in these systems and an approach to the optimization of these systems. The theory shall be applied to the following well-known file systems: multilist files, inverted files, and a descriptive storage proposed by Wong and Chiang.

We shall model (section 4.2) a multilist file and inverted file as a "generalized file structure", proposed by Hsaio and Harary and later extended by Manola and Hsaio. We shall demonstrate how the retrieval algorithm of Hsaio and Harary may be generalized by the application of the theory of f-covers. The original algorithm, proposed by these authors, is shown to be a special case of our general algorithm.

The multilist file is treated (section 4.2.1) as an example of the generalized file structure. Hence, the general algorithm developed in section 4.2 applies to the multilist file. By introducing a restriction on the implementation of this file system (i.e., ordering lists of records according to their relative addresses) we derive a new algorithm called the generalized trace algorithm. This algorithm, which may be applied to any Boolean query appropriate to a multilist file,

is shown to be an effective method of retrieval for this file system. For a given Boolean query, any f-cover for the query may be used as the expression to which the algorithm is applied. An analysis of the effectiveness of several such f-covers is presented. In addition, we present an alternative method of implementing the multilist file.

A second special case of a generalized file structure is the inverted file (section 4.2.2). With regard to this file type, we shall discuss the problems of implementing the general algorithm proposed in section 4.2. We shall develop several analytic results concerning the optimal collection of inverted lists, when the format of the queries is restricted. We shall demonstrate the difficulty of determining analytic results when the format of the queries is unrestricted. Hence, we shall conclude that an appropriate optimization technique is a search procedure such as the B&B approach proposed in chapter three. The effectiveness of this approach shall be demonstrated by referencing a case study (Appendix D).

Wong and Chiang have proposed a descriptive-storage system which we shall abbreviate as W&C descriptive storage. The authors restrict the definition of sets to a special format, given the collection of attributes which generates the collection of sets. We shall demonstrate (section 4.3) how the theory of f-covers generalizes retrieval using a W&C descriptive storage. We shall discuss the problem of determining the optimal collection of attributes (from which

to generate the W&C descriptive storage) and we shall demonstrate how the optimal collection may be determined by adapting the B&B method.

Lastly (section 4.4), we shall propose for future research an alternative approach to the optimization of STDS. File systems which use this approach we shall call "work-set" systems because of their similarity with paging systems. We shall indicate a method for optimizing such systems and recommend that future research be directed to analyzing this approach.

4.2 Generalized File Structure

In this section we shall discuss a particular type of Set-theoretical Descriptive Storage, called a generalized file structure. This formulation was first proposed by Hsiao and Harary [HSIAO70] and later extended by Manola and Hsiao [MAN73]. Several well-known file structures, such as inverted files and multilist files, may be defined as special cases of the generalized file structure.

We shall develop the definition of a generalized file structure according to Hsiao and Harary. These authors indicate that retrieval may be accomplished according to an algorithm called the general retrieval algorithm. We shall formulate this algorithm in our own terms and we shall indicate how it may be improved. The improvement involves determining an optimal f-cover for a particular query. Lastly we shall discuss the extensions proposed by Manola and Hsiao. We shall indicate how the results that we have derived with regard to the original definition also apply to this extension.

We shall first present some preliminary definitions. Consider two sets A (attributes) and V (values) of undefined elements. A record R is a subset of the Cartesian product A$\times$V where each attribute has exactly one value. Some of the attribute-value pairs are called keywords. The index of a record is the set of keywords that characterize the record. Every record has associated with it a unique positive integer called its address. Sometimes a record R has associated with

one of its keywords K the address of another record containing the keyword K. We call this address a K-pointer of R. When there is no address associated with a keyword K in a record R, we say P has a null K-pointer. A mechanized record has a pointer (either null or indicating another record) assocated with all keywords in its index. Henceforth, when we speak of a record, we shall mean a mechanized record.

A K-list L is a set of records containing a keyword K such that

4.2-1 (i) the K-pointers for the records are distinct

(ii) each non-null K-pointer in L indicates a record within L

(iii) there is a unique record in L, called the beginning of the list, which is not pointed at by any K-pointer in L

(iv) there is a unique record in L, called the end of the list, which has a null K-pointer.

A file F of records is a set of records in which every K-list containing one or more records in F. Files are identified by unique names called file names.

A directory of a file F is a collection of sequences

4.2-2 $(K_i, n_i, h_i; a_1^i, a_2^i, \ldots, a_{h_i}^i)$ where

(i) $1 \le i \le m$

(ii) there are m keywords

(iii) n_i is the number of keywords containing keyword K_i.

(iv) h_i is the number of K_i-lists.

(v) a_j^i ($1 \leq j \leq h_i$) is the beginning of the j-th K_i-list.

A generalized file structure is a file with its directory. The generalized file structure may be used to define several well-known file structures by placing restrictions upon the directory as follows:

4.2-3
(i) Inverted file: $n_i = h_i$
(ii) Index Random: $n_i = h_i = 1$
(iii) Index Sequential: $n_i = h_i = 1$, $a_1^1 < a_1^2 < \ldots < a_1^m$
(iv) Multilist file: $h_i = 1$

Diagrams 4.2-4 and 4.2-5 schematically depict the inverted-list and multilist file organizations, respectively. Keywords are represented in these diagrams by capital letters. The records are shown as boxes with their addresses to the top left corner. Inside the boxes the K-pointers are shown and the symbol $\emptyset$ is used for the null pointer.

Hsiao and Harary suggest that an algorithm, known as the general retrieval algorithm, be used to determine all records which satisfy a given query. A keyword K is said to be true for a given record R if R contains K. A query is a Boolean expression of keywords and a query is true for a record R, if this proposition is true for R. In this case we say R satisfies the query. It is convenient to define several functions before specifying the general retrieval algorithm.

The directory-search function f is used to select the

4.2-4 Inverted File

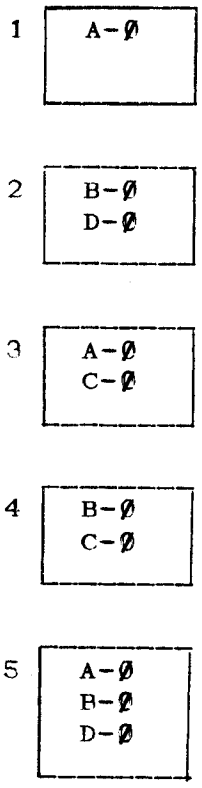
Directory <A,3,3;1,3,5>

<B,3,3;2,4,5>

<C,2,2;3,4>

<D,2,2;2,5>

File



4.2-5 Multilist File

Directory

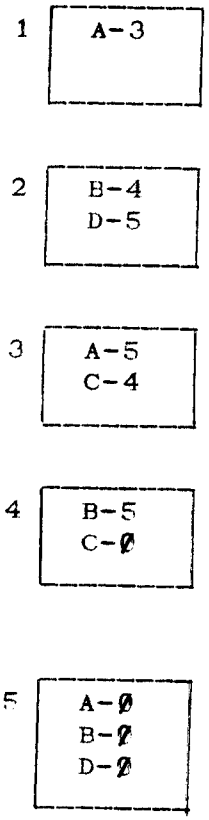
<A,3,1;1>

<B,3,1;2>

<C,2,1;3>

<D,2,1;2>

File



beginning of the K-lists, in ascending order according to the addresses of the records:

4.2-6 $$f(K_i,x) = \min\{a^i_j \mid 1\le j\le h_i\} \quad \text{if } x=0;$$
$$= \min\{a^i_j \mid a^i_j > x \text{ and } 1\le j\le h_i\} \quad \text{otherwise.}$$

when the directory entry for keyword K_i is 4.2-2.

The file-search function g is used to trace the records which occur on an arbitrary K-list:

4.2-7 Let x be the address of a record containg a keyword K. If this record has a null K-pointer, then $g(K,x)=0$. Otherwise, $g(K,x)$ is the address of the record specified by the K-pointer.

The general retrieval algorithm may be specified in six steps as follows:

4.2-8 (i) Choose the prime keywords: Write the query in the form:

$$(K_1^1 \wedge K_2^1 \wedge \cdots \wedge K^1) \vee \cdots \vee (K_1^s \wedge K_2^s \wedge \cdots \wedge K^s_{r(s)})$$

Denote by K'_i the keyword which has fewest records associated with it, with respect to the i-th term. Hence, for the s items in the query, we have $t\le s$ prime keywords which we denote $K'_1, K'_2, \ldots, K'_t$.

(ii) Search the Directory: Create a sequence Φ by repeatedly applying f to the directory entries of K'_i $(1\le i\le t)$. Then, Φ has $h = \sum_{i=1}^{t} h'_i$ addresses of

lists. Let E be an empty set of records.

(iii) Search the file: Removing duplicates, let G be the collection of addresses in Φ. If all the records specified by the addresses in G have been retrieved, then go to step (vi). Otherwise choose the least address a' in G of a record which has not yet been retrieved. Retrieve the record R at address a'. Let H be the collection of addresses $g(K_i', a')$ $(1 \leq i \leq t)$.

(iv) Examine the record: If record R satisfies the query, then add R to the set E.

(v) Prepare to retrieve the next record: Move all the addresses in H to Φ. Repeat the algorithm at step (iii).

(vi) Complete the algorithm: The collection E of records contains all records which satisfy the query.

Example 4.2-9 illustrates the general retrieval algorithm for the multilist file 4.2-5 and the query $(A \wedge D) \vee (A \wedge C)$.

Hsiao and Harary indicate that the general retrieval algorithm is significant because it retrieves records at most once and because it attempts to minimize the number of records retrieved at step (iii). The authors call steps (ii) - (vi) parallel processing of lists. It is evident that the first step of the algorithm selects an f-cover

4.2-9 General retrieval algorithm for multilist file

query	$(A \wedge D) \vee (A \wedge C)$
step (i)	prime keywords are D,C
step (ii)	$\Phi = (2,3)$; $E = \emptyset$
step (iii)	G = (2,3) ; a'=2; H=(5)
step (iv)	record (2) does not satisfy query
step (v)	$\Phi = (2,3,5)$
step (iii)	G = (2,3,5); a'=3; H=(4)
step (iv)	add record (3) to E: E = {3}
step (v)	$\Phi = (2,3,5,4)$
step (iii)	G = (2,3,4,5); a'=4; $H=\emptyset$
step (iv)	record (4) does not satisfy the query
step (v)	$\Phi = (2,3,4,5)$
step (iii)	G = (2,3,5,4); a'=5; $H=\emptyset$
step (iv)	add record (5) to E: E = {3,5}
step (v)	$\Phi = (2,3,5,4)$
step (iii)	G = (2,3,5,4); go to step (vi)
step (vi)	stop

4.2-10 $q' = K'_1 \vee K'_2 \vee \cdots \vee K'_t$

and the remaining steps retrieve all records which satisfy the expression 4.2-10. The authors note that the algorithm is not optimal in that it may retrieve more records than are to be included in the retrieval set E. We claim that the algorithm is also not optimal with regard to the f-cover 4.2-10 selected.

We believe that the criteria by which an f-cover should be selected is the expected number of records to be retrieved. Suppose there are n records in a file in question and that parallel processing of lists is used to determine records which satisfy a query

4.2-11 $q = K_1 \vee K_2 \vee \cdots \vee K_t$

Then, the probability that an arbitrary record has a keyword K_i is given by

4.2-12 $p_i = n_i/n$

Assuming that the keywords are distributed independently of one another, the probability that an arbitrary record satisfies the query 4.2-11 is given by

4.2-13 $P(q) = 1 - \prod_{i=1}^{t}(1-p_i)$

i.e., the probability that an arbitrary record is retrieved by parallel processing of lists acting on the query q is specified by P(q).

We propose that step (i) of the general retrieval algorithm be changed so that an f-cover q' of the form 4.2-10 be chosen, such that P(q') is minimum. This may be accomplished as follows:

4.2-14 (i) Select an optimal f-cover: Write the query q in the form $q=(K_1^1 \vee K_2^1 \vee \cdots \vee K_r^1)_{(1)} \wedge \cdots \wedge (K_1^s \vee K_2^s \vee \cdots \vee K_r^s)_{(s)}$ and chose as an optimal f-cover q' the term for which $P(K_1^i \vee K_2^i \vee \cdots \vee K_r^i)_{(i)}$ is minimum.

(ii) Retrieve the records which satisfy the f-cover in (i) and add to the retrieval set E all records for which q is true.

Referring to example 4.2-9, the query may be written as

4.2-15 $q = A \wedge (D \vee E)$

and so we have

4.2-16 $P(A) = 1 - (1-.6) = .6$

$P(D \vee E) = 1 - (1-.4)\cdot(1-.4) = .64$

In this case, q'=A would be the optimal f-cover and three records, as compared to four, would be retrieved by parallel processing of lists.

The analysis in this section has disregarded the costs of determining the best f-cover and of manipulating the directory. The cost of determining the best f-cover may be neglected if this cost is relatively small, compared to the

cost of retrieval. It is therefore important that the determination of optimal f-covers be implemented efficiently. The determination of optimal f-covers is concerned with the definitions (and file statistics) and not with the contents of lists. Hence, the pointers to the beginning of lists should be stored separately from the remainder of the directory entries, if a given directory entry contains relatively many pointers. This generalization, called the separation principle, was discussed in section 2.2.

Manipulation of record pointers, obtained from the pertinent directory entries, becomes increasingly costly as the number of pointers per directory entry increases. The two extremes are multilist (one pointer per directory entry) and inverted file (all non-null pointers are found in the directory). In the case where there are many pointers per directory entry, this cost of manipulation must be considered during the determination of the optimal f-cover. Thus, in section 4.2-2 we consider this factor when we discuss the creation of optimal f-covers for inverted files.

Manola and Hsiao [MAN73] have extended the definition of generalized file structure by introducing the concept of secondary-storage cells or cells. These entities are intended to model the physical characteristics of direct-access storage devices such as moving-head disks. An integer s is called the cell size and the cell number c of a record with address a is calculated as

4.2-18 $$c = \lfloor a/s \rfloor$$

The authors consider directory entries to be sequences

4.2-19 $(K_i, n_i, h_i; (c_1^i, a_1^i, p_1^i), (c_2^i, a_2^i, p_2^i), \ldots, (c_{h(i)}^i, a_{h(i)}^i, p_{h(i)}^i))$

where the start of a K_i-list is indicated by a

4.2-20 (c_j^i, a_j^i, p_j^i) such that:

(i) a_j^i is the address of the beginning of this K_i-list

(ii) c_j^i is the cell number of the beginning of this K_i-list

(iii) p_j^i is the number of records in this K_i-list.

All K-lists are restricted so that any list contains only records having the same cell number. With these extensions, some well-known file structures are defined as follows (n is the number of records in the file).

4.2-21 (i) Multilist: $h_i=1$, $s=n$

(ii) Cellular multilist: The c_j^i are distinct within a directory entry.

(iii) Inverted-list: $s=1$, $n_i = h_i$, $p = 1$

(iv) Index Random: $n_i = h_i = 1$

(v) Index Sequential: $n_i = h_i = 1$, $a_1^1 < a_1^2 < \ldots < a_1^m$

The authors present an algorithm, for this extended definition, which corresponds to the general retrieval

algorithm defined by Hsiao and Harary. This algorithm may be summarized as follows:

4.2-22 (i) For a query in the form $(K_1^1 \wedge K_2^1 \wedge \ldots \wedge K_{r(1)}^1) \vee \ldots \vee (K_1^t \wedge K_2^t \wedge \ldots \wedge K_{r(t)}^t)$ define

(a) θ_j^i to be the cell addresses in the directory entries of keyword K_j^i.

(b) $\theta_i = \bigcap_{j=1}^{r(i)} \theta_j^i$, $1 \leq i \leq t$

(ii) For every $c \in \bigcup_{i=1}^{t} \theta_i$

(a) create an f-cover $q'(c) = K^1 \vee K^2 \vee \ldots \vee K^t$.

(b) use parallel processing of lists to retrieve all records, in the storage cell with number c, which satisfy q'(c). Examine these records and add to the retrieval set all records for which q is true.

Thus, the algorithm has essentially two parts:

4.2-23 (i) Select the smallest collection of storage cells which may possibly contain a record for which the query is true.

(ii) For each cell in (i) retrieve a superset of those records for which the query is true. By examination of these records, select those records for which the query is true.

Our analysis of retrieval in the generalized file structure defined by Hsiao and Harary will apply in (ii) of 4.2-23 since

the algorithm 4.2-22 treats each storage cell as a file in the original sense. Because the generalized file structure includes several well-known file systems, these results have wide applicability. Similar remarks will apply to the results we shall obtain in the subsections 4.2.1 and 4.2.2. These two subsections are concerned with two particular examples of the generalized file structure: multilist files and inverted files.

4.2.1 Multilist File

In this subsection we shall be concerned with the multilist file. We have already indicated (in section 4.2) a method of retrieval for this file, by improving the general retrieval algorithm of Hsiao and Harary. By adding a restriction to the way lists are constructed, we may specify a more general algorithm for retrieval, called the generalized trace algorithm. This algorithm will retrieve using any f-cover. When the f-cover is in the form required by the general retrieval algorithm, the generalized trace algorithm is equivalent to parallel processing of lists. In addition, we shall propose a new way of implementing the multilist file and we shall show some conditions under which it is preferable to the traditional method of implementation.

Further optimization of retrieval using a multilist file is possible if the following restriction is placed upon every K-list: (records stored at consecutive integral addresses)

4.2.1-1 For every K-list containing a record with address x and with a K-pointer to a record with address y, $x < y$.

In other words, if we were to trace any list, the records would be encountered in ascending order, according to their addresses. When this condition is true, parallel processing of lists is equivalent to the following algorithm:

4.2.1-2 (i) Let the f-cover to be traced be $q=K_1 \vee K_2 \vee \cdots \vee K_s$. For $1 \leq i \leq s$ associate a pointer p_i

with the i-th keyword K_i and let the initial value of p_i be the address of the first record in the list for keyword K_i. Let E be an empty set of records.

(ii) Let $x = \min\{p_1, p_2, \ldots, p_s\}$. If $a > n$ (n is the maximum address of a record in the multilist file), then continue at step (v). Retrieve the record at address x. If the keyword K_i ($i \leq i \leq s$) occurs on this record, then

(a) if the K_i-pointer is null, let p_i have the value n+1;

(b) otherwise, let p_i have the value of the K_i-pointer.

(iii) Examine the record. If the query is true for the record, then add the record to the set E.

(iv) Repeat the algorithm at step (ii).

(v) Complete the algorithm: the set E contains exactly those records for which the query is true.

Several features of this algorithm are interesting. In the first case, if a keyword K_i ($1 \leq i \leq s$) was found in the f-cover, and if no list was defined for it, then every record in the file has to be examined. We could accomplish this if we initially gave p_i the value 1 and set p_i to (x+1) whenever step (ii) of the algorithm was performed. In the second case, if the f-cover was of the form

4.2.1-3 $q = K_1 \wedge K_2 \wedge \bullet\bullet\bullet \wedge K_s$

then we could retrieve fewer records by selecting (step (ii)) the next address x as

4.2.1-4 $x = \max\{p_1, p_2, \bullet\bullet\bullet, p_s\}$

These two considerations introduce the concept of a trace function.

Informally, the trace function operates upon the f-cover, in conjunction with the pointers associated with keywords in the f-cover, to produce the address of the next record to be retrieved. The recursive definition of the trace function J is as follows:

4.2.1-5: Let p(K) be the pointer associated with a keyword K, let E_1, E_2 be Boolean expressions of keywords, and let x be the address of the last record retrieved.

(i) $J(0,x) = x+1$

(ii) $J(1,x) = x+1$

(iii) $J(K,x) = \max\{p(K), x+1\}$

(iv) $J(\bar{K},x) = x+1$

(v) $J(E_1 \vee E_2, x) = \min\{J(E_1,x), J(E_2,x)\}$

(vi) $J(E_1 \wedge E_2, x) = \max\{J(E_1,x), J(E_2,x)\}$

The above definition applies to f-covers in which complementation is applied only to keywords. We shall develop

our analyses using f-covers in this restrictive format. We shall subsequently generalize our results to all expressions. Example 4.2.1-7 shows the trace function for several queries.

We have developed the trace function in order to implement an efficient and general algorithm, called the trace algorithm. This algorithm repeatedly uses the trace function to determine the address of the next record for which a query (in the restricted format) could be true. We specify the trace algorithm as follows:

4.2.1-6 (i) Select an f-cover: For an arbitrary query, q_0, construct an appropriate f-cover q. Let the keywords in q be $\{K_1, K_2, \ldots, K_t\}$. For $1 \leq i \leq t$ associate a pointer p_i with the keyword K_i and let p_i have value of -1.

(ii) Initialize the pointer variables: For $1 \leq i \leq t$ if a directory entry $(K_i, n_i, 1; a_i)$ exists for a keyword K_i, then let p_i have a value a_i. Let a variable x have a value of 0 and let E be an empty set of records.

(iii) Search the file: Let $y = J(q,x)$. If $y > n$ (n is the maximum address of a record in the file) then continue at step (vi). Retrieve the record R with address y. If $K_i \in R$ $(1 \leq i \leq t)$ then let p_i have a value of n+1 if the K_i-pointer is null. If the K_i-pointer is not null then let p_i have the value of that pointer.

(iv) Examine the record: If the query q_0 is true for the record R, then add R to the set E.

(v) Prepare to retrieve the next record: Set the value of x to y. Repeat the algorithm at step (iii).

(vi) Complete the algorithm: The set E contains exactly those records which satisfy the query q_0.

Example 4.2.1-8 illustrates the trace algorithm applied to the query (A∧C) ∨ (A∧D) for the multilist file 4.2-5. Values of the various variables are presented at the end of each step. In this example three records were read. If the algorithm of Hsiao and Harary were used, then four records would have been read.

We shall now demonstrate that the trace algorithm determines all records which satisfy a given f-cover (in the restricted format). As we shall show, the value of a pointer associated with a given keyword is either

4.2.1-9 (i) less than the address of the last record retrieved, or

(ii) equal to the address of the next record, following the last record retrieved, for which the associated keyword is true.

We shall call this property x-consistency of pointers, where x is the address of the last record retrieved. As a

4.2.1-7 Examples of Trace Functions

Query	Trace Function
$q = K_1 \vee K_2$	$J[q,x] = \min\{ J[K_1,x], J[K_2,x] \}$ $= \min\{ \max\{p_1,x+1\}, \max\{p_2,x+1\} \}$ $= \max\{ x+1, \min\{p_1,p_2\} \}$
$p = K_1 \wedge K_2$	$J[q,x] = \max\{ J[K_1,x], J[K_2,x] \}$ $= \max\{ \max\{p_1,x+1\}, \max\{p_2,x+1\} \}$ $= \max\{ x+1, p_1, p_2 \}$
$q = (K_1 \vee K_2) \wedge (K_1 \vee K_3)$	$J[q,x] = \max\{ J[K_1 \vee K_2,x], J[K_1 \vee K_3,x] \}$ $= \max\{ \max\{ x+1, \min\{p_1,p_2\} \},$ $\max\{ x+1, \min\}p_1,p_3\} \} \}$ $= \max\{ x+1, \min\{p_1,p_2\}, \min\{p_1,p_3\} \}$
$q = (\bar{K}_1 \wedge K_2) \vee (K_1 \wedge \bar{K}_3)$	$J[q,x] = \min\{ \max\{x+1,\max\{x+1,p_2\} \},$ $\max\{x+1,\max\}x+1,p_1\} \} \}$ $= \max\{ x+1, \min\{p_1,p_2\} \}$

where p_1, p_2, p_3 are the pointers associated with the keywords K_1, K_2, K_3 respectively.

Example 4.2.1-8: Trace algorithm

$$q = q_0 = (A \wedge C) \vee (A \wedge D)$$

$$J[q,x] = \max\{\min\{\max\{p(A),x+1\},\max\{p(C),x+1\}\},$$
$$\min\{\max\{p(A),x+1\},\max\{p(D),x+1\}\}\}$$
$$= \max\{\, x+1,\ p(A),\ \min\{p(C),p(D)\}\ \}$$

STEP	x	y	p(A)	p(C)	p(D)	E
i	-	-	-1	-1	-1	ø
ii	0	-	1	3	2	ø
iii	0	2	1	3	5	ø
iv	0	2	1	3	5	ø
iii	2	3	5	3	5	ø
iv	2	3	5	4	5	{3}
iii	3	5	6	4	6	{3}
iv	3	5	6	4	6	{3,5}
iii	5	6	6	4	6	{3,5}

vi and the algorithm terminates

convenience, we first define a function m(q,x) as follows:

4.2.1-10 Definition: For a multilist file, let E be the collection of record addresses for records which satisfy a query q. Then, $m(q,x) = \min \{y \mid y \in E_0 \{n+1\}$ and $y>x\}$ for $0 \leq x \leq n$ where n is the maximum address of a record in the file.

For the multilist file 4.2-6, we have

4.2.1-11 $m(A \vee C, x) = 1, \quad x=0$

$= 3, \quad x=1,2$

$= 4, \quad x=3$

$= 5, \quad x=4$

$= 6, \quad x=5$

We may now formally define x-consistency as follows:

4.2.1-12 Definition: For a file of records in which n is the maximum address of a record and for an integer x ($0 \leq x \leq n$), a pointer p(k) associated with a keyword K is x-consistent if and only if one of the following is true:

(i) $p(K) < x$

(ii) $p(K) = m(K,x)$

If all pointers are x-consistent, then the trace function indicates a record address greater than x and less than or equal to the address of the next record which can satisfy the query in the trace function:

4.2.1-13 Lemma: For an arbitrary query q (in the restricted format), if all pointers associated with keywords occurring in q are x-consistent, then $x < J(q,x) \leq m(q,x)$.

proof: We shall prove the lemma by using induction on a measure of complexity c(q) defined for Boolean expressions of keywords. Let c(q) be the number of occurrances of variables and constants in the expression q (i.e., $c(A \vee B \vee C \vee A) = 4$).

Basis Step: Suppose c(q)=1. Then q is either a constant, an uncomplemented keyword, or a complemented keyword. In the first and third cases the hypothesis is true because $J(q,x) = x+1$. In the second case the hypothesis is true by definitions of J and of x-consistency.

Induction Step: Suppose c(q)>1. Assume by induction that the hypothesis is true for all expressions e which satisfy c(q)>c(e). Now q may be expressed in one of the following formats:

(i) $q = e_1 \vee e_2$

(ii) $q = e_1 \wedge e_2$

where e_1 and e_2 are expressions such that $c(q) > c(e_1)$ and $c(q) > c(e_2)$.

case(i): $q=e_1 \vee e_2$

$m(q,x) = \min\{m(e_1,x), m(e_2,x)\}$ because q is true for the record with address $\min\{m(e_1,x), m(e_2,x)\}$ and because no record with an address y where $x<y<\min\{m(e_1,x), m(e_2,x)\}$ can satisfy q without contradicting the definitions $m(e_1,x)$ and $m(e_2,x)$. Now, since $c(q)>c(e_1)$ and $c(q)>c(e_2)$ we have $x<J(e_1,x)\leq m(e_1,x)$ and $x<J(e_2,x)\leq m(e_2,x)$. So we derive:

$x < \min\{J(e_1,x), J(e_2,x)\} \leq \min\{m(e_1,x), m(e_2,x)\}$ or

$x < J(q,x) \leq m(q,x)$

which completes case (i) of the proof.

case(ii): $q = e_1 \wedge e_2$

The proof of this case is omitted as it is proven similarily to case (i). Thus, for all cases, $x < J(q,x) \leq m(q,x)$ which completes the proof of this lemma.

Lemma 4.2.1-13 provides the basis for the proof that the trace algorithm retrieves all records that satisfy an arbitrary query (in the restricted format), provided that we can demonstrate that x-consistency of pointers is maintained. Consider a pointer p, associated with a keyword K, just after a record with address x is retrieved. If p is x-consistent, then one of the following is true:

4.2.1-14

(i) $p < x$

(ii) $x < p < J(q,x)$

(iii) $p = J(q,x)$

(iv) $J(q,x) < p$

} $p = m(k,x)$

for some query q.

Suppose that the next record to be retrieved has an address y=J(q,x). The value of p is changed if and only if K occurs on the record with address y. An examination of the four cases above indicates that, after updating the pointers, the pointers are y-consistent. We can now show the following lemma:

4.2.1-15 Lemma: For an arbitrary query q (in the restricted format) the trace algorithm will retrieve all records for which the query is true.

Consider the case where at least one record satisfies the query. Let Z_1 be the least address of a record for which the query is true. All pointers associated with keywords in q are initially 0-consistent and so that first record retrieved (with address r_1) satisfies

4.2.1-16 $0 < r_1 \leq m(q,0)$ or
$0 < r_1 \leq Z_1$

The pointers are updated and become r_1-consistent. If $r_1 \neq Z_1$, then the second record retrieved (with address r_2) satisfies

4.2.1-17 $r_1 < r_2 \leq m(q,r_1)$ or
$r_1 < r_2 \leq Z_1$

In general, after t retrievals, if $r_1, r_2, \ldots, r_t \neq Z_1$, then

we have

4.2.1-18 $0<r_1<r_2<\cdots<r_t\leq Z_1$

This process must be finite because

4.2.1-19 (i) The algorithm terminates after t retrievals if and only if $J(q,r_t) > n$.

(ii) There are only a finite sequences 4.2.1-18 possible.

Hence, there must exist a t such $r_t = Z_1$; i.e., the first record is retrieved. By similar arguments, we can show that the record with address Z_i, where

4.2.1-20 Z_i = min{y| record with address y satisfies q and y>Z }

is retrieved, given that the record Z_{i-1} is retrieved.

We shall now indicate how the trace function can be generalized to operate upon all Boolean expressions of keywords. Table 4.2.1-21 is the complete definition of the trace function J(q,x,c) where q and x are as before and where c is a modulus-two counter of the depth of complementation of a sub-expression. This function is illustrated in example 4.2.1-22. For the Boolean expressions E, E_1, E_2, the following identities are true:

4.2.1-23 (i) $\overline{(\bar{E})} = E$

(ii) $\overline{(E_1 \vee E_2)} = \bar{E}_1 \wedge \bar{E}_2$

Table 4.2.1-21 Generalized Trace Function

Let p(k) be the pointer associated with a keyword K. Let E_1, F_1, E_2 be Boolean expressions of keywords.

$$J[K,x,0] = \max\{p(K),x+1\}$$

$$J[K,x,1] = x+1$$

$$J[\bar{K},x,0] = x+1$$

$$J[\bar{K},x,1] = \max\{p(K),x+1\}$$

$$J[F_1 \vee E_2,x,0] = \min\{ J[E_1,x,0], J[E_2,x,0] \}$$

$$J[E_1 \vee E_2,x,1] = \max\{ J[E_1,x,1], J[E_2,x,1] \}$$

$$J[E_1 \wedge E_2,x,0] = \max\{ J[E_1,x,0], J[E_2,x,0] \}$$

$$J[E_1 \wedge E_2,x,1] = \min\{ J[E_1,x,1], J[E_2,x,1] \}$$

$$J[\bar{E},x,0] = J[E,x,1]$$

$$J[\bar{E},x,1] = J[E,x,0]$$

$$J[1,x,0] = x+1$$

$$J[1,x,1] = x+1$$

$$J[0,x,0] = x+1$$

$$J[0,x,1] = x+1$$

where n = the maximum address of a record in the file.

Example 4.2.1-22 Generalized Trace Function

$$J[\overline{(\bar{K}_1 \vee \bar{K}_2) \wedge (\bar{K}_1 \vee \bar{K}_3)}, x, 0]$$

$$= J[(\bar{K}_1 \vee \bar{K}_2) \wedge (\bar{K}_1 \vee \bar{K}_3), x, 1]$$

$$= \min\{J[(\bar{K}_1 \vee \bar{K}_2), x, 1], J[(\bar{K}_1 \vee \bar{K}_3), x, 1]\}$$

$$= \min\{ \max\{J[\bar{K}_1, x, 1], J[\bar{K}_2, x, 1]\},$$
$$\max\{J[\bar{K}_1, x, 1], J[\bar{K}_3, x, 1]\} \}$$

$$= \min\{ \max\{ \max[x+1, p_1],$$
$$\max[x+1, p_2] \},$$
$$\max\{ \max[x+1, p_1],$$
$$\max[x+1, p_3] \} \}$$

$$= \min\{ \max\{x+1, p_1, p_2\}, \max\{x+1, p_1, p_3\} \}$$

$$= \max\{ x+1, p_1, \min\{p_2, p_3\} \}$$

where p_1, p_2, p_3 are the pointers associated with the keywords K_1, K_2, K_3 respectively.

(iii) $\overline{(\bar{E}_1 \wedge \bar{E}_2)} = \bar{E}_1 \vee \bar{E}_2$

Repeated application of these three identities will reduce any query to the restricted format described above. The function of the parameter c is to simulate such a reduction process.

Using the generalized form of the trace function, the following lemmas may be proven:

4.2.1-24 Lemma: For an arbitrary query q, if all pointers in q are x-consistent, then $x < J(q,x,0) \leq m(q,x)$ and $x < J(q,x,1) \leq m(q,x)$.

4.2.1-25 Lemma: For an arbitrary query q, the generalized trace algorithm will retrieve all records for which the query is true.

Lemma 4.2.1-24 is proven similarily to lemma 4.2.1-13 except that the measure of complexity c(q) for a Boolean expression q is defined to be the sum of the number of times variables, constants, and operators occur in the expression. In addition, there is a third case ($q=\bar{e}$) to be considered during the induction step. Lemma 4.21-25 is proven identically to lemma 4.2.1-15, except that lemma 4.2.1-24 is cited in place of 4.2.1-13.

We now define the generalized trace algorithm as the trace algorithm (4.2.1-6) with the generalized trace function used in place of the trace function. In multilist files in which the lists are ordered according to the values of the pointers, the generalized trace algorithm is an attractive

method for a number of reasons. The method is simple to program and is very general as it operates on any query. In addition, the query may be used in its original form with the following interpretations:

4.2.1-26 (i) Keywords K are interpretted as integers with a value max {p(k),x+1} where x is the address of the last record retrieved (or zero).

(ii) 'AND' and 'OR' are interpretted as 'max' and 'min' according to the formulas in table 4.2.1-21.

(iii) The complementation operator has the effect of adding one (modulo two) to the third parameter of the trace function.

The generality of the trace algorithm allows all possible f-covers to be used. We propose that an attempt be made to select the f-cover for which the expected number of records to be retrieved is least. We shall indicate how probabilistic analysis can be applied in a number of cases.

We shall present a comparative analysis of several cases. The general problem of determining the expected number of records retrieved for an arbitrary f-cover has not been solved. In our analyses we shall assume that the probability that an arbitrary word contains a specified keyword is known. For a file of n records, m of which contain a given keyword, the corresponding probability may be calculated as the ratio

m/n. We shall further assume that the keywords are distributed among the records of the file independently of one another.

The first case to be considered is where the query is of the form

4.2.1-27 $q = K_1 \vee K_2 \vee \cdots \vee K_s$

and where $p_1, p_2, \cdots, p_s$ are the probabilities associated with s keywords, respectively. As may be verified, the generalized trace algorithm, applied to q, retrieves exactly those records which contain one or more of the s keywords. The probability that an arbitrary record contains at least one of the s keywords is given by

4.2.1-28 $$1 - \prod_{i=1}^{s} (1-p_i)$$

provided that all the s keywords have lists defined for them. In the case where at least one of the keywords does not have a list defined to correspond to it, all records in the multilist file are retrieved. Summarizing these results, we derive:

4.2.1-29 $r(q) = 1 - \prod_{i=1}^{s} (1-p_i)$, if all of the s keywords have lists defined

$= 1$, if at least one of the s keywords does not have a list defined.

as the probability that an arbitrary record is retrieved when the generalized trace algorithm is applied to a query in the form 4.2.1-27. The optimal f-cover, in this restricted

format, is the one for which 4.2.1-29 is least.

The second case which we shall consider is where the query is in the form:

4.2.1-30 $q = K_1 \wedge K_2 \wedge \bullet\bullet\bullet \wedge K_s$

In this situation, the trace function is defined as

4.2.1-31 $J(q,x,0) = \max \{x+1, p_1, p_2, \bullet\bullet\bullet p_s\}$

We shall view the generalized trace algorithm as a system of state transitions where

4.2.1-32 (i) the state of the system is determined by the set of keywords for a record which is retrieved; and

(ii) a transition occurs whenever a record is retrieved.

We shall specify a method to compute the probability that the system is in a given state. An optimal f-cover, in the format 4.2.1-30, is one for which the probability that the system is in the state consisting of all keywords is greatest. We compute this probability by modelling our state-transition system as a Markov chain.

4.2.1-33 Definition: Consider r transitions in a system with n states. Let E (i,j) denote the event that the i-th state occured as a result of the j-th transition. Then, the system is a Markov chain if

for any integer k ($1 \le k \le r$) and for any integers j_1, j_2, ..., j_k ($1 \le j_i \le n, 1 \le i \le k$), the states $E(1,j_1)$, $E(2,j_2)$, ..., $E(k,j_k)$ satisfy $P[E(k,j_k)|E(k-1,j_{k-1}), E(k-2,_{k-2}), \ldots, E(1,j_1)] = P[E(k,j_k)|E(k-1,j_{k-1})]$.

In other words, a Markov chain is a system where the k-th transition, from state i to a state j, depends only upon the states i,j and the time of the transition. If the transition probabilities are independent of time, then the system is said to be homogeneous. When we speak of a Markov chain henceforth, we mean a homogeneous Markov chain. We denote the probability of a transition form state i to a state j by $P(i,j)$ and we denote by $P_m(i)$ the probability the system is in a state i after m transitions. The Markov chain is said to be ergodic if the numbers $\Pi(i)$ ($1 \le i \le n$) exist such that

4.2.1-34 $$\lim_{m \to \infty} P_m(i) = \Pi(i)$$

The numbers $\Pi(i)$ are called the stationary probabilities of the Markov chain. Two states are said to communicate if one may be reached from the other, and vice versa. The following lemma [FFLL50] is well-known.

4.2.1-35 If all states in a Markov chain communicate and if a state i exists such that $P(i,i) > 0$, then the Markov chain is ergodic.

The stationary probabilities satisfy

4.2.1-36: $$\Pi(j) = \sum_{k=1}^{n} \Pi(k) \cdot P(k,j)$$

$$\Pi(j) > 0$$

$$1 = \sum_{k=1}^{n} \Pi(k)$$

In the system we have specified, all states (except the state corresponding to no keywords) communicate since records with any collection of keywords may be retrieved. Hence, we may determine the stationary probabilities.

Of course our system is not strictly a Markov chain because the probabilities of transition from one state to another is dependent upon the records which have been previously retrieved. We shall ignore this effect in our analysis. The calculation of the transition probabilities shall consider only the probabilities of the keywords occuring on an arbitrary record. Thus, our calculations are based upon 'sampling with replacement' when 'sampling without replacement' should be used. In the limit, as the number of records in the file is increased unbounded, the two methods of sampling become identical. Hence, for large files, the difference between the actual and computed values will be small. Appendix E gives the derivation of the transition probability from a state corresponding to the keywords

4.2.1-37 $K_1, K_2, \cdots, K_v, K_{v+1}, K_{v+2}, \cdots, K_s$

to a state corresponding to the keywords

4.2.1-38 $K_1, K_2, \cdots, K_v, K_{s+1}, K_{s+2}, \cdots, K_{s+u}$

when the query is of the form

4.2.1-39 $q = K_1 \wedge K_2 \wedge \cdots \wedge K_t$

We define $\Phi(P,j)$ to be the collection of subsets of P which have exactly j elements of P. Then, the required probability is given by the formula.

4.2.1-40: $$\left[\prod_{i=1}^{v} p_i\right]\left[\prod_{i=v+1}^{s} (1-p_i)\right]\left[\prod_{i=s+1}^{s+u} p_i\right]\left[\prod_{i=s+u+1}^{t} (1-p_i)\right]\left[L(P_1) - L(P_2) \right]$$

where $P_1 = \{p_1, p_2, \cdots, p_s\}$

$P_2 = \{p_{v+1}, p_{v+2}, \cdots, p_s\}$

$$L(P) = \sum_{i=1}^{k} (-1)^{i+1} \sum_{Y \in \Phi(P,i)} \left[\frac{1}{1-\prod_{p \in Y}(1-p)}\right]$$

and P has k elements.

A transition between two states whose associated keyword collections are disjoint is impossible and so the transition probability will be zero.

As an example, consider the query

4.2.1-41 $q = A \wedge B$

where the capital letters denote keywords. The transition probabilities are displayed in table 4.2.1-44. Small letters, corresponding to the keywords, are used to denote the probabilities that the respective keywords occur on an arbitrary record. The steady-state equations, corresponding to equations 4.2.1-36, are shown in diagram 4.2.1-45. The solutions for these equations are shown in diagram 4.2.1-46.

4.2.1-44: Transition probabilities for 4.2.1-41.

TRANSITION	PROBABILITY
{A,B} to {A,B}	$a \cdot b \cdot \left[\frac{1}{a} + \frac{1}{b} - \frac{1}{1-(1-a)\cdot(1-b)}\right]$
{A,B} to {A}	$a \cdot (1-b) \cdot \left[\frac{1}{a} - \frac{1}{1-(1-a)\cdot(1-b)}\right]$
{A,B} to {B}	$(1-a) \cdot b \cdot \left[\frac{1}{b} - \frac{1}{1-(1-a)\cdot(1-b)}\right]$
{A} to {A,B}	$\frac{a \cdot b}{a}$
{A} to {A}	$\frac{a \cdot (1-b)}{a}$
{A} to {B}	0
{B} to {A,B}	$\frac{a \cdot b}{b}$
{B} to {A}	0
{B} to {B}	$\frac{(1-a) \cdot b}{b}$

Table 4.2.1-45: Steady-state equations for 4.2.1-41

α = probability that system is in state {A,B}

β = probability that system is in state {A}

γ = probability that system is in state {B}

$$\alpha = a \bullet b \bullet \left[\frac{1}{a} + \frac{1}{b} - \frac{1}{1 - (1-a) \bullet (1-b)}\right] \bullet \alpha + \frac{a \bullet b \bullet \beta}{a} + \frac{a \bullet b \bullet \gamma}{b}$$

$$\beta = a \bullet (1-b) \bullet \left[\frac{1}{a} - \frac{1}{1 - (1-a) \bullet (1-b)}\right] \bullet \alpha + \frac{a \bullet (1-b) \bullet \beta}{a}$$

$$\gamma = (1-a) \bullet b \bullet \left[\frac{1}{b} - \frac{1}{1 - (1-a) \bullet (1-b)}\right] \bullet \alpha + \frac{(1-a) \bullet b \bullet \gamma}{b}$$

Diagram 4.2.1-46: Stationary probabilities for 4.2.1-41

$$\alpha = \frac{1 - (1-a) \bullet (1-b)}{1 + (1-a) \bullet (1-b)}$$

$$\beta = \gamma = \frac{(1-a) \bullet (1-b)}{1 + (1-a) \bullet (1-b)}$$

Of particular interest is the probability for the state corresponding to both attributes:

4.2.1-42 $$\Pi(A,B) = \frac{1 - (1-a)\cdot(1-b)}{1 + (1-a)\cdot(1-b)}$$

As an alternative to 4.2.1-41, the f-cover

4.2.1-43 $$q(A) = A$$

could be used in the generalized trace algorithm. In this situation the expectation that a record with the keywords A, B is retrieved is given by

4.2.1-48 $$\Pi(A) = b$$

Then, the expression 4.2.1-41 is preferred to 4.2.1-43 if

4.2.1-49 $$\Pi(A) < \Pi(A,B)$$

or if

4.2.1-50 $$b < \frac{1-(1-a)\cdot(1-b)}{1+(1-a)\cdot(1-b)}$$

or if

4.2.1-51 $a > b/(1+b)$ when $b \neq 1$.

In the case where b=1, the relationship 4.2.1-50 is always false and so the condition 4.2.1-49 is false. These results are summarized by the diagram 4.2.1-52. A unit square is divided into three areas. Each of the areas where one of the queries q, q(A), q(B) is preferred to the others is labelled.

A coordinate (a,b) within the square represents the probabilities for the keywords A and B, respectively. Thus, the query q = A ∧ B is preferred when the associated probabilities (a,b) fall within the area delineated by the lines

4.2.1-53 b = 1

a = 1

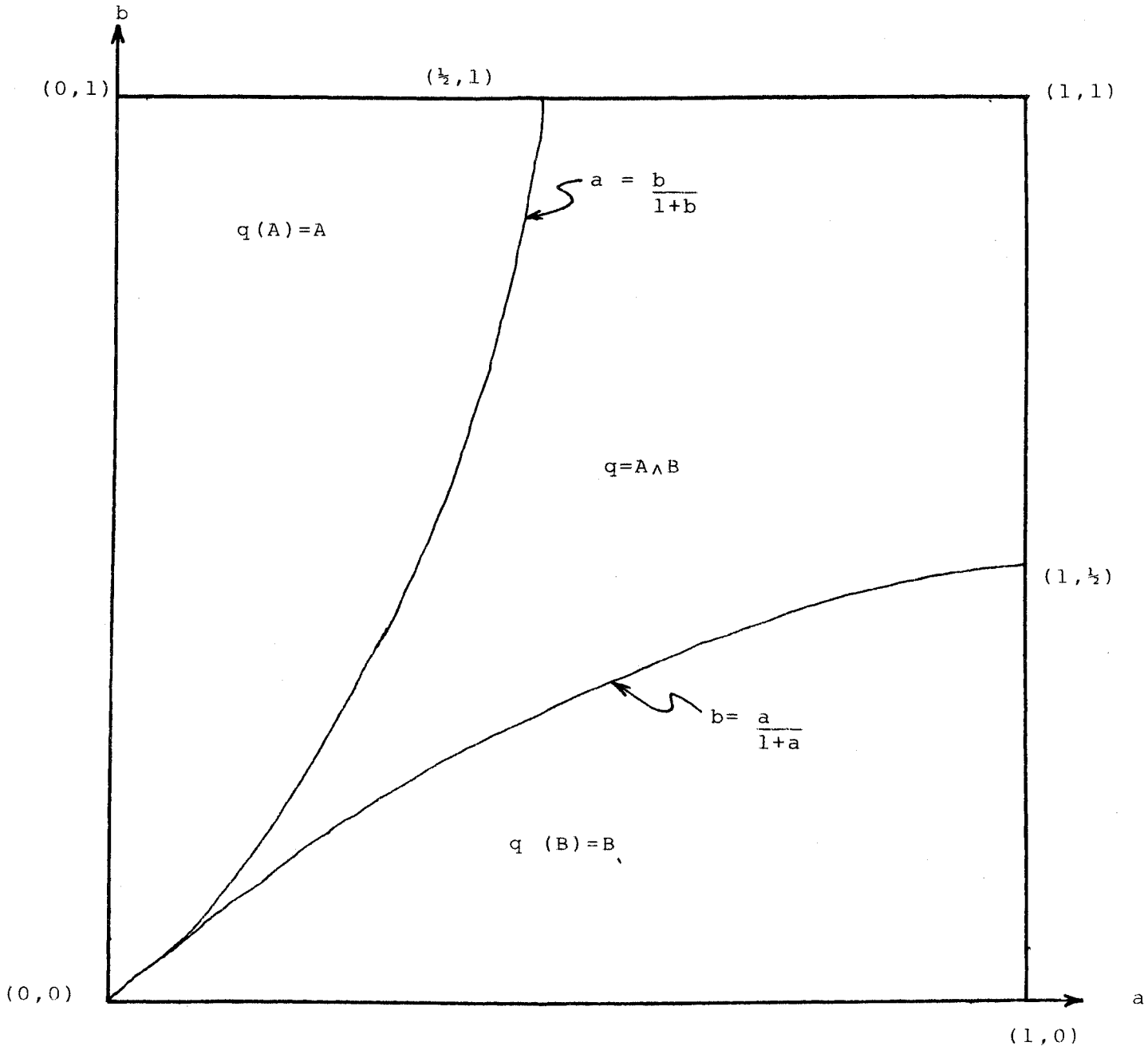
a = b/(1+b)

b = a/(1+a)

It is conceded that the transition-system method is impractical in many cases. For s keywords in a query 2^s-1 linear equations are generated. This number of equations may be too large for practical purposes. For the case of s=2, the equations were solved analyticly and the results summarized in diagram 4.2.1-52. For larger values of s, the analytic solution of the equations resulted in formulas which were to complex to be useful.

Lastly, we shall propose a new method of implementing a multilist file. According to Hsaio and Harary, a multilist file is a general file structure in which the directory entry for any keyword has exactly one list associated with it. Each list is represented as a collection of records in which the next record in the list is indicated by a K-pointer. These pointers are usually stored as part of the record from which they point. We suggest that these pointers be stored 'parallel' to these records; i.e., that another file be created to consist of the K-pointers. Thus, if the i-th

4.2.1-52: Optimal areas for f-covers of A ∧ B



record in the multilist file has the keywords K_1, K_2, ..., K_t associated with it and if lists are defined for each of these keywords, then the i-th descriptive record in the parallel file would be represented as

4.2.1-54 $\{<K_1,p_1>,<K_2,p_2>,....,<K_t,p_t>\}$

Another possibility is to invert each of these descriptive records as

4.2.1-55 $\{<p_1,K_1^1,K_2^1,...,K_{n(1)}^1>, ..., <p_s,K_1^s,K_2^s,...,K_{n(s)}^s>\}$

when there are n(i) ($1 \leq i \leq t$) keywords having the pointer value p_i. The representation 4.2.1-55 may use less storage space than 4.2.1-54 but will be less efficient to use to locate the pointer value corresponding to an arbitrary keyword.

The proposed implementation will decrease retrieval costs in some cases. Consider the generalized trace algorithm applied to a multilist file which is implemented in this manner. The descriptive records may be stored in blocks and so a single access to direct-access storage may read into the computer memory several descriptive records. In some cases, the descriptive record by itself may determine whether or not an actual record satisfies a query. Thus, there is no need to read the actual data record and evaluate the query expression for it. It is anticipated that many retrievals can be performed with substantially fewer accesses using the proposed method.

In order to illustrate how a record may be evaluated as

true or false, with regard to a query, consider the query

4.2.1-56 $q = (A \vee B) \wedge C$

where A, B, C represent keywords. If a descriptive word is read in which A, C are keywords (as indicated by the keyword pointers), then the query is true for that record. In the case where the keyword C is not found on a descriptive record, and a list is defined for that keyword, then the record cannot satisfy the record. If lists exist for keywords A, C (and not for B) and if C is found on the record and A is not, then the actual record has to be examined (the query will be true if keyword B is true for the record).

In general, the keywords in a query may be divided into two disjoint classes: those for which lists are defined, and those for which lists are not defined. By examining the descriptive record, all keywords in the former set may be evaluated. Substituting the appropriate values into the query expression and applying the identities

4.2.1-57 (i) $K = K \vee 0$

(ii) $K = K \wedge 1$

(iii) $0 = K \wedge 0$

(iv) $1 = K \vee 1$

(for a keyword K)

we may reduce the query to either a constant or an expression involving keywords from the latter class. When the reduced expression is a constant, then it has been determined whether

or not the query is true for the record in question. When the substitutions do not reduce the query to a constant, the actual record must be read to determine whether or not the record satisfies the query. Of course, if all the keywords in a query have lists defined to correspond to them, then the query expression can always be reduced to a constant (since all keywords may be evaluated by inspecting the descriptive record). As an example, consider the query 4.2.1-56 and suppose that keywords A, C have lists defined. Then, the following contingency table indicates the values of A and C for which the actual record must be read in order to evaluate 4.2.1-56.

Table 4.2.1-58

A \ C	0	1
0	0	?
1	0	1

("?" indicates when $q = (A \vee B) \wedge C$ cannot be evaluated without examination of the actual record.)

In summary, this subsection has been concerned in particular with certain improvements to the multilist file organization. Specifically, we introduced a restriction which may be applied to the lists and then we showed how the retrieval process could be generalized using the modified structure. We derived several results concerning the

probability of one f-cover being preferred to another. Lastly, we proposed a new manner of implementing a multilist file and indicated the potential benefits from such an implementation.

4.2.2 Inverted file

This subsection will be concerned with inverted files. We shall consider inverted files from the same point of view as Hsaio and Harary. Parallel-processing of lists for this particular type of generalized file structure will be reviewed and compared with an algorithm by Manola and Hsiao The two algorithms above are special cases of a general algorithm which we shall specify. We shall consider an inverted file in which the queries are restricted to a simple format and develop an analytic result indicating how to determine the optimal collection of inverted lists for this system. Lastly, we shall illustrate why analytic results are difficult to obtain for systems in which the queries are unrestricted. In the absence of analytic results, a search procedure such as the branch-and-bound approach is indicated and we shall reference a case study to illustrate our ideas.

Adopting the view of Hsiao and Harary, an inverted file is a generalized file structure for which the directory entries are (n=h):

4.2.2-1 (K,n,h;a(1),a(2),...,a(h))

Thus, all records in the file which contain the keyword K are indicated by a pointer a(i) ($1 \leq i \leq h$) in the associated directory. It is common to call 4.2.2-1 an inverted list and, since all K-pointers associated with records are null, to omit physically representing K-pointers.

Parallel-processing of lists, applied to the inverted file, is equivalent to the following algorithm:

4.2.2-2 (i) Search the directory: Create a sequence θ to be the union of the record pointers found in the directory entries (4.2.2-1) for the prime keywords K_i ($1 \le i \le t$). Let E be an initially empty set of records.

(ii) Search the file: If θ is null, then go to step (v). Otherwise, let $a' = \min\{a \mid a \in \theta\}$. Retrieve the record R at address a'.

(iii) Examine the record: If the query is true for record R, then add R to set E.

(iv) Prepare to retrieve the next record: Delete (all occurances of) a' from θ. Repeat the algorithm at step (ii).

(v) Complete the algorithm: The collection E of records contains all records which satisfy the query.

Manola and Hsaio suggest that the following algorithm be applied to inverted lists:

4.2.2-3 (i) For a query q in the form $q=(K_1^1 \wedge K_2^1 \wedge \cdots \wedge K_{r(1)}^1) \vee \cdots \vee (K_1^s \wedge K_2^s \wedge \cdots \wedge K_{r(s)}^s)$ create a collection of sets θ_i ($1 \le i \le S$) as $\theta_i = \bigcap_{j=1}^{r(i)} \theta_j^i$ where θ_j^i is the collection of

record pointers in the directory entry (4.2.2-1) for keyword K_j^i in q.

(ii) Let $\theta = \bigcup_{i=1}^{s} \theta_i$ and let E be an initially empty set of records.

(iii) - (vi) are the same as steps (ii) - (v) of algorithm 4.2.2-2, except that the query q is always true for the record R retrieved and so does not have to be examined.

We consider both algorithms to be special cases of the general algorithm:

4.2.2-4 (i) Create an f-cover: Given a query q_0, create an appropriate f-cover q such that q involves no complementation and such that all keywords have lists defined to correspond to them.

(ii) Create θ: Create a set θ of record pointers defined by substituting (simultaneously) in q as follows:

(a) substitute '∩', '∪' for '∧', '∨' respectively

(b) substitute $\theta(K)$ for K, where $\theta(K)$ is the collection of record pointers which are found in the directory entry for a keyword K in q.

(iii) Retrieve the records: Retrieve all records indicated by the pointers in θ. Add to the retrieval set those records for which the query q_0 is true.

Clearly, algorithm 4.2.2-2 selects an f-cover in the form of the 'OR' of prime keywords. In algorithm 4.2.2-3, the initial query q_0 is used as the f-cover.

There may be, however, a number of other f-covers which can be selected. It is reasonable to attempt to select the f-cover for which the cost of retrieval will be minimum. To estimate this cost we have to have knowledge of the retrieval algorithm(s) and accounting procedures. As accounting procedures, for example, typically differ from computer system to computer system, analytic results which generally apply are few. We anticipate, however, given the definitions of retrieval algorithms and accounting procedures which are applicable to a particular system, estimates of retrieval costs may be made. To illustrate the estimation of these costs we shall consider a simple accounting system and a simple retrieval algorithm.

Consider an accounting system where the cost of performing a retrieval is the sum of

4.2.2-5 (i) the cost of retrieving records: assumed to be proportional to the number of records retrieved.

(ii) the cost of taking the intersections and unions of lists of pointers: assumed to be proportional to the number of such operations.

For an f-cover q, the cost c(q) of retrieval is therefore estimated to be:

4.2.2-6 $c(q) = X \cdot a(q) + Y \cdot b(q)$ where

(i) $a(q)$ is the number of records retrieved.

(ii) X is the expected cost of retrieving a record and examining it to determine if it satisfies the query.

(iii) $b(q)$ is the number of intersections or unions required to create the set θ in algorithm 4.2.2-4.

(iv) Y is the average cost of taking the union or intersection of two sets of record pointers.

Now, for an f-cover q in the form

4.2.2-7 $$q = (K_1^1 \vee K_2^1 \cdots \vee K_{n(1)}^1) \wedge (K_1^2 \vee K_2^2 \cdots \vee K_{n(2)}^2) \wedge \cdots \wedge (K_1^s \vee K_2^s \vee \cdots \vee K_{n(s)}^s)$$

the number of set operations $b(q)$ is given by

4.2.2-8 $$b(q) = \sum_{i=1}^{s} n(i) - 1$$

if all operations are binary. To estimate the number of records $a(q)$ to be retrieved, we first compute the probability $p(q)$ that an arbitrary record satisfies the f-cover q. We assume that we know, for any keyword K, the probability $p(K)$ that an arbitrary record contains K. $p(q)$ may be estimated as follows:

4.2.2-9 (i) write the f-cover q in disjunctive normal form.

(ii) create an expression by (simultaneously) performing the following substitutions into the Boolean expression (i):

(a) '•' for '∧'

(b) '+' for '$_{\vee}$'

(c) p(K) for a keyword K•

(d) (1-p(K)) for $\bar{K}$

(iii) evaluate (ii) to derive p(q)•

Method 4•2•2-9 derives p(q), under the assumptions above, because the method finds the sum of the probabilities that a record is a member of the disjoint sets defined by the disjunctive terms in (i) of 4•2•2-9• For example, the f-cover

4•2•2-10 $q = (A \wedge B) \vee C$

is written in disjunctive normal form as

4•2•2-11 $q = (A \wedge B \wedge C)$
$\vee \quad (A \wedge B \wedge \bar{C})$
$\vee \quad (A \wedge \bar{B} \wedge C)$
$\vee \quad (\bar{A} \wedge B \wedge C)$
$\vee \quad (\bar{A} \wedge \bar{B} \wedge C)$

and the estimate of p(q) is given by

4•2•2-12 $p(q) = p(A) \bullet p(B) \bullet p(C)$
$+ \; p(A) \bullet p(B) \bullet (1-p(C))$

$$+\ p(A)\cdot(1-p(B))\cdot p(C)$$

$$+\ (1-p(A))\cdot p(B)\cdot p(C)$$

$$+\ (1-p(A))\cdot(1-p(B))\cdot p(C)$$

Having calculated p(q), the estimate a(q) is computed as

4.2.2-13 $a(q) = N\cdot p(q)$

where N is the number of records in the file.

It remains to specify the retrieval algorithm. We shall assume that the query q_0 is in the form

4.2.2-14 $$q_0 = (K_1^1 \vee K_2^1 \vee \cdots \vee K^1_{n(1)}) \wedge (K_1^2 \vee K_2^2 \vee \cdots \vee K^2_{n(2)}) \wedge \cdots \wedge (K_1^s \vee K_2^s \vee \cdots \vee K^s_{n(3)})$$

One of the s terms in 4.2.2-14, for which the n(i) ($1 \le i \le s$) keywords all have inverted lists defined to correspond to them, is selected as the f-cover. We shall choose the f-cover for which cost is estimated to be minimum. As an example, consider the query.

4.2.2-15 $q_0 = (A \vee B \vee C) \wedge (A \vee D \vee E) \wedge (A \vee F)$

where the keywords are represented as capital letters and where the following probabilities are known:

4.2.2-16 $p(A) = .3$

$p(B) = .2$

$p(C) = .4$

$p(D) = .1$

$p(E) = .3$

$p(F) = .6$

The remaining parameters to be specified are as follows:

4.3.2-17 $X = 50$

$Y = 500$

$N = 1{,}000$

The following f-covers may be selected

4.3.2-18 $q_1 = A \vee B \vee C$

$q_2 = A \vee D \vee E$

$q_3 = A \vee F$

We can calculate

4.3.2-19 $a(q_1) = 664 \quad b(q_1) = 2$

$a(q_2) = 559 \quad b(q_2) = 2$

$a(q_3) = 720 \quad b(q_3) = 1$

$c(q_1) = 50 \times 664 + 500 \times 2 = 4320$

$c(q_2) = 50 \times 559 + 500 \times 2 = 3795$

$c(q_3) = 50 \times 720 + 500 \times 1 = 4100$

And so the f-covers are preferred in the order: q_2, q_3, q_1. If the cost of the set operations, Y, was increased to 1000 then q_3 would be the preferred f-cover.

The preceding analysis has shown that the cost of an f-

cover is dependent upon a number of factors. It is not possible, in most cases, to choose the f-cover having least expected cost without considering all these factors. A somewhat similar situation occurs if we attempt to determine the best collection of lists to define.

Let us consider a system in which the queries have the very simple form:

4.2.2-20 $q = K$

where K is a keyword. We shall assume that the following parameters are known:

4.2.2-21 (i) $\{K_1, K_2, \ldots, K_n\}$ is the set of keywords which may be used in the queries

(ii) C_i is the cost of performing the retrieval for the i-th query when the i-th keyword has a list defined for it.

(iii) S is the cost of sequentially processing the file to determine all records which satisfy an arbitrary query.

(iv) L_i is the cost of storing the list for K_i during a period of time.

(v) f_i is the expected frequency which the i-th query is entered during the period of time.

We shall consider K_i to be a bivalent variable with a value of

one if the i-th keyword has a list defined for it and a value of zero otherwise. The expected system cost, for the unit of time, is a function of the n variables K_i and is given as

4.2.2-22 $$C(K_1,K_2,\ldots,K_n) = B + \sum_{i=1}^{n} \{ K_i \bullet L_i + f_i \bullet (K_i \bullet C_i + (1-K_i) \bullet S)\}$$

where B is the cost to store the basic information for the period of time. The cost function is minimized, if for $1 \leq i \leq n$,

4.2.2-23 $$K_i = 1, \text{ if } f_i \bullet S \geq L_i + f_i \bullet C_i$$
$$= 0, \text{ otherwise}$$

In other words, we define a list for the i-th keyword if the relationship

4.2.2-24 $$f_i > L_i / (S-C_i)$$

is true. The above relationship indicates that, even for this simple system, the decision whether or not to define an inverted list is dependent upon four factors. The situation is more complicated when the queries which characterize system usage are allowed to be more complex.

Consider one such query,

4.2.2-25 $$q = (A \vee B) \wedge (B \vee C) \wedge (C \vee D)$$

involving the keywords A, B, C, D. Suppose that the retrieval algorithm will choose one of

4.2.2-26 $$q(1) = A \vee B$$
$$q(2) = B \vee C$$

$$q(3) = C \vee D$$

as the f-cover to be used, in, for example, algorithm 4.2.2-4, and assume that the f-covers are preferred in ascending order, according to their indices. In this case, the cost of retrieval for this query is given by

4.2.2-27 $$c(q) = ABR_1 + (1-A)BCR_2 + (1-A)(1-B)CDR_3 + (1-A)(1-B)(1-C)S$$

where A, B, C, D are variables as before and where R_1, R_2, R_3 are the costs of performing retrievals using the three f-covers respectively. S is the cost of sequentially searching the file.

If there were many similarly complex queries, than the cost equation for the system cannot be minimized as easily as was 4.2.2-15.

This analysis indicates the need for a procedure which will effectively search for an optimal (or nearly optimal) solution. As was stated in chapter three, the branch-and-bound method is an attractive method to use for this purpose. Appendix D is a case study of an inverted file to which this method was applied.

In summary, our general approach using f-covers has increased the number of possible ways to retrieve records using an inverted file. Some of these ways may decrease retrieval cost substantially, compared to conventional approaches. The analysis, to determine an optimal f-cover for

a particular query, is naturally dependent upon the retrieval algorithms and accounting procedures applicable in a particular system. We have indicated how these factors may be taken into consideration in a few specific cases. The important feature is the approach used. We anticipate that this method has wide applicability.

We have indicated that analytic results are, in general, difficult to determine. As an alternative, we have proposed that the branch-and-bound technique be adapted as a search procedure to determine the (nearly) optimal collection of inverted lists. A case study has been provided to illustrate the application of these techniques (Appendix D).

4.3 Wong-and-Chiang Descriptive Storage

In this section we shall consider an organization of descriptive storage proposed by Wong and Chiang [WONG71]. This organization may be shown to be a special case of Set-theoretical Descriptive Storage. We shall demonstrate how retrieval may be generalized for the Wong and Chiang (W&C) organization. Lastly, we shall illustrate how this form of descriptive storage may be optimized, using the branch-and-bound technique.

The authors use definitions similar to those of Hsaio and Harary to formulate their model. A record is considered to be a finite collection of attribute-value pairs. Each record has a unique positive integer associated with it, called its address. A certain sub-collection of these attribute-value pairs are called keywords. Descriptive storage is organized as follows. Let $A = \{K_1, K_2, \ldots, K_n\}$ be the keywords defined for a file G. Then there are 2^n expressions of the form

4.3-1 $\quad \mathcal{K}_1 \wedge \mathcal{K}_2 \wedge \ldots \wedge \mathcal{K}_n$

where $\mathcal{K}_i$ represents either K_i or $\bar{K}_i$ $(1 \leq i \leq n)$.

Descriptive storage consists of those non-null sets of records which satisfy a definition of the form 4.3-1. As is shown by the authors, every record occurs in exactly one such set. This organization is clearly an STDS $\langle G, A, D, \mathcal{S} \rangle$ where basic storage is the original file G, the set A is a set of keywords (attributes in our earlier definitions), and $\mathcal{S}$ is the collection of sets defined above. The set-definition function

4.3.4 Wong-and-Chiang file organization

FILE

Address	Keywords
1	A, C
2	A, B
3	D
4	A, C
5	A, C, D
6	D
7	A, B
8	A, C, D
9	D
10	A, B

DESCRIPTIVE STORAGE

Definition	Contents
$\bar{A} \wedge \bar{B} \wedge \bar{C} \wedge D$	{3, 6, 9}
$A \wedge \bar{B} \wedge C \wedge \bar{D}$	{1, 4}
$A \wedge \bar{B} \wedge C \wedge D$	{5, 8}
$A \wedge B \wedge \bar{C} \wedge \bar{D}$	{2, 7, 10}

4.3.5 (i) Let $A=\{K_1, K_2, \ldots, K_n\}$ be the keywords for which the W&C descriptive storage is constructed. For an arbitrary query q and for the set of keywords, construct the minimal f-cover q'.

(ii) Write q' in disjunctive normal form. For every term in this expression, if there exists a set having the term as its definition, retrieve and examine all records in the set. For every record which satisfies the original query q, add the record to the retrieval set.

In the case where all the attribute-value pairs in a query are keywords, the minimal f-cover is equivalent to the original expression. There is an advantage in using f-covers which are minimal because:

4.3-6 (i) The number of records for which an f-cover of q (with respect to A) is true, is minimum for the minimal f-cover.

(ii) Exactly those records, for which the f-cover is true, are retrieved by the algorithm.

In other words, if the f-cover is not minimal, then more records than needed may be retrieved. Thus, we have generalized the retrieval algorithm to handle any query which may be posed.

For an arbitrary file there are many ways of constructing

a W&C descriptive storage because there are many ways to specify a set A. This set, in conjunction with the contents of the file, determines the descriptive storage. The optimal collection of keywords is one for which the system cost is expected to be minimum. In the preceding chapters we have discussed the factors which influence system cost. As an illustration of how to determine the optimal descriptive storage for the W&C organization, we shall assume that the cost C(A,q) of performing a retrieval for a query q is given by

4.3-7 $$C(A,q) = r(A,q) \cdot R + C(q)$$

where (i) r(A,q) is the number of records retrieved and examined during algorithm 4.3-5.

(ii) R is the (constant) cost to read and examine a record.

(iii) C(q) is the remaining (constant) cost of retrieval when the query is q.

We shall characterize expected system cost, in the usual way, by the sets Q (queries) and F (frequencies). Then, the expected system cost C(A), when the set A of keywords generates descriptive storage, is given by

4.3-8 $$C(A) = S(A) + \sum_{q \in Q} f(q) \cdot C(A,q)$$

where (i) S(A) is the cost of storing the sets in descriptive storage, for an arbitrary period of

It remains to specify how to derive a lower bound for expected cost of a set of feasible solutions during the branch-and-bound optimization. Recall that a subset is characterized as a four-tuple

4.3-15 $\langle j, D, Cmin, x \rangle$ where

(i) for n candidates, $0 \leq j \leq n$

(ii) $D \subseteq \{K_1, K_2, \bullet\bullet\bullet, K_j\}$

(iii) Cmin is a lower bound on cost, when the descriptive storage is generated by $D \vee D'$, where $D' \subseteq \{K_{j+1}, K_{j+2}, \bullet\bullet\bullet, K_n\}$.

(iv) x is a symbol indicating whether or not the subset of solutions is feasible.

Given j and D in 4.3-15, we compute a lower bound for cost as

4.3-16 $$Cmin = S(D) + \sum_{q \in Q} f(q) \bullet C(D'', q)$$

where (i) $D'' = D \cup \{K_{j+1}, K_{j+2}, \bullet\bullet\bullet, K_n\}$

(ii) S(D) is the expected cost to store the descriptive storage generated by the collection of keywords D.

Having completely defined all computations, we may determine the optimal W&C descriptive storage, using the branch-and-bound algorithm specified in chapter three.

In summary, we have applied our general results regarding Set-theoretrical Descriptive Storage to the particular case of a Wong-and-Chiang descriptive storage. By doing so, we have

4.4 Work-Set Systems

In this section we shall be concerned with a special class of STDS systems which we shall call work-set systems. We have borrowed the term 'work-set' from paging systems because of the manner in which our proposed systems will decide upon the composition of descriptive storage. We shall treat these systems as proposals for further research. We shall indicate the potential benefits of such systems and we shall present some elementary results regarding their operation.

First we shall explain the reason for naming these systems 'work-set systems'. Consider the execution of a computer program in a paging system. The program and data areas of the program are partitioned into small portions called pages. For a given number of machine instructions, only a few of the data and program pages are referenced. This collection is called the working set or work-set for the series of instructions in question. Pages that are not in the work-set need not reside in the computer memory while the sequence of instructions is executed by the computer hardware. Thus, by writing unneeded pages from the computer memory to a direct-access storage device, and by reading required pages from a direct-access storage device into the computer memory, the program may be executed using less memory (at any given time) then the total requirements of the program. It is argued that paging is desireable because

4.4-1 (i) It is possible to execute programs whose total memory requirements exceed the physical memory of the computer.

(ii) It is possible to allow more programs to execute concurrently (because more will fit into the physical memory of the computer) and so the hardware resources of a computer system are more effectively utilized.

Paging systems attempt to estimate the size and contents of the work-set for a given program. Algorithms, known as replacement algorithms, determine where a page of a program is to reside - in the computer memory or on a direct-access storage device. Decisions on whether to transfer a given page from one memory hierarchy to another are typically based upon:

4.4-2 (i) the historical reference pattern of a program
(ii) the environment of the paging system
(iii) the current state (i.e., ready to execute, waiting for a resource, etc.) of the program.

The goal of the paging system is to maximize the usage of the system resources, subject to constraints such as "response time for terminal users must not exceed some maximum value".

We propose that an STDS system can be implemented in a manner analogous to a paging system. The various sets in an STDS system may be considered in the same manner as pages are in a paging system. Corresponding to replacement algorithms

in the paging systems, we propose that algorithms which decide to create or destroy sets be implemented. Based upon the usage of the system, these decisions would control which sets consititute descriptive storage. This collection of sets resembles the collection of pages in the work-set. A comparison between paging systems and the proposed STDS system is displayed in table 4.4-3.

Thus the essential feature of the work-set systems is that an optimal descriptive storage is approximated, during the system operation, through the action of replacement algorithms. These algorithms decide

4.4-4 (i) to destroy a set when the storage of the set (for a period of time) is expected to cost more than the saving in retrieval cost achieved by its presence.

(ii) to create a set when storage of this set is expected to cost less than the saving in retrieval cost due to its presence.

Replacement algorithms, in paging systems, must be invoked frequently (i.e., many times a second) and must be completed in a short period of time. Otherwise, the system overhead becomes excessive and various measures of system performance (such as response time for terminal users) will indicate inadequate usage of system resources. In the proposed work-set systems, however, the rate at which queries are posed to the system determines that replacement decisions are made

relatively infrequently (i.e., on the order of hours or days). As a result a relatively more complex analysis may be performed.

As an example, consider the simple inverted file outlined in section 4.2.2 where queries have the simple form:

4.4-5 $q = K$

where K is a keyword.

As was shown, a list should be defined for a keyword K if the following relationship is true:

4.4-6 $$f(K) > \frac{L(K)}{S-C(K)}$$

where (i) f(K) is the frequency which the query 4.4-5 is entered (for a period of time).

(ii) L(K) is the cost of storing the inverted list for keyword K (for the period of time).

(iii) S is the cost of retrieval when basic storage is sequentially processed.

(iv) C(K) is the cost of retrieval for query 4.4-5 when the indicated inverted list is defined.

The three variables on the right side of the greater-than sign are immediately available and may be combined to yield a lower bound

4.4-7 $$F(K) = \frac{L(K)}{S-C(K)}$$

The set in question should be defined if the difference (4.4-9) is greater than zero. If we substitute the mean cost C for C_i ($1 \leq i \leq n$) we obtain the expression

4.4-10 $$\text{Difference} = \sum_{i=1}^{n} f_i \cdot (S-C) - L$$

which indicates that the set in question should be defined if the relationship

4.4-11 $$\sum_{i=1}^{n} f_i > \frac{L}{S-C}$$

is true. For n=1, the relationship 4.4-11 is identical to 4.4-6. Again, the right-hand side of the relationship is constant and the problem reduces to estimation of f_i ($1 \leq i \leq n$).

The above two cases are easily solved because the total cost of the system can be expressed as a sum of functions, each of which involves a single bi-valent variable representing whether or not a set is defined. In the latter case above, the total cost of the system may be expressed as

4.4-12 $$\text{Cost} = \sum_{j=1}^{m} D(l_j) = \sum_{j=1}^{m} \left[S \cdot (1-l_j) \cdot \sum_{i=1}^{n(j)} f_i^j + l_j \cdot \{ L(l_j) + \sum_{i=1}^{n(j)} f_i^j \cdot C_i \} \right]$$

where (i) l_j ($1 \leq j \leq m$) is a variable whose value is one if the j-th list is defined, and zero otherwise.

(ii) $D(l_j)$ ($1 \leq j \leq m$) is the cost 'contribution' for queries which can be satisfied using the j-th set.

(iii) S is the (constant) cost to determine the retrieval set for an arbitrary query by sequentially processing the file.

(iv) f_i^j $(1 \leq i \leq n(j),\ 1 \leq j \leq m)$ is the frequency which the i-th query corresponding to the j-th set is entered.

(v) C_i^j $(1 \leq i \leq n(j),\ 1 \leq j \leq m)$ is the cost of performing a retrieval for the i-th query of the j-th set, when that set is defined.

(vi) $L(l_j)$ $(1 \leq j \leq m)$ is the cost to store the j-th set.

This situation is not possible when the more general case exists; i.e., if

4.4-13 (i) several sets may be combined to yield a superset of the retrieval set for a query; and

(ii) a query may be processed using several collections as in (i).

In this general situation the psuedo-Boolean cost equation will involve terms where several of the bi-valent variables are multiplied together. Hence, the equation is not separable as was done in the restricted cases.

It is proposed that a function be introduced which measures the 'usefulness' of a given set. For the two special cases cited above, suitable functions might be:

4.4-14 case (1): $f(K) - \frac{L(K)}{S-C(K)}$

4.5 Summary

This chapter was intended to emphasize the applicability of our results to particular file systems. We have also presented several results which apply to particular file systems.

The significance of our work may be summarized as follows:

4.5-1 (i) The retrieval of records which satisfy a given Boolean query has been generalized to include all appropriate queries. The introduction of f-covers has considerably expanded the choice of retrieval algorithms, since many f-covers may exist for an arbitrary query.

(ii) The optimization of descriptive storage has been attempted from a general point of view. Previously, it was either not attempted or it was performed only for particular systems in restricted environments.

a significant step towards the automatic optimization of storage structures.

Several avenues for future research are indicated. Concerning the generation of f-covers, two aspects to be investigated are immediately apparent. In many systems, the disjunctive normal form of a query will be inconvenient to work with. We need a method to generate f-covers which does not involve using the query in disjunctive normal form. Secondly, in some cases the special knowledge of the system may indicate that particular types of f-covers are likely to have the minimum cost. For a system which generates a number of f-covers and then chooses the best, the generation could be done in a heuristic manner. In order to determine the feasibility of this approach we need some experiments with a real-life system.

In the area of optimization of systems, we have provided a foundation. A general method has been proposed and a number of implementation considerations have been discussed. We have indicated the theoretical feasibility of the approach. To demonstrate the practical feasibility of the approach, experience with existing systems is necessary. It is expected that heuristics will be developed (i.e., better branching rules) which will decrease the actual number of steps in the optimization algorithm, based upon special knowledge about particular systems.

It is anticipated that the results of this thesis can be applied in other areas. For roughly the past five years the

idea of a relational data base has been gaining increasing acceptance. Based upon the theoretical ideas of Childs [CHILD67A, CHILD67B], Codd [CODD70, CODD71], and others [ABR70, DATE71A, DATE71B, EARL71, HEATH71], several examples of relational data bases [BRAC72] have been implemented. It is not known whether or not these systems employ descriptive storage to a large degree. In any case, the idea of optimizing these systems has not yet been considered. It is anticipated that these systems may be optimized somewhat similarly to STDS systems. The branch-and-bound method, for example, might be used to decide which derived relations should be maintained in the data base.

Lastly, we have proposed an alternative approach to determine the contents of descriptive storage. Because of an analogy with paging systems, we call these systems "work-set" systems. We recommend that these systems be investigated.

In conclusion, we feel that the results of this research are significant and will contribute to the implementation of data base and file systems in an economical manner.

APPENDIX (A): DEFINITIONS:

An *accounting formula* is a function which, for a given collection of measurements of system resources, assigns a cost.

An *accounting procedure* is a method by which cost is assigned to the usage of computer resources. Typically, the usage of system resources is measured and an accounting formula is applied to these measurements in order to determine the cost.

Sometimes every record in a file has a unique (within the file) integer, called an *address*, associated with it. Examples of addresses include the physical location (on a hardware device) of the record and the relative record number of the record within a file.

In a file system, *basic storage* is that information which is of primary interest to the users (to be distinguished from descriptive storage, whose only function is to facilitate access to the basic information).

A variable is *bivalent* if it can have one of exactly two possible values.

A *Boolean expression* is defined recursively as follows:

(i) 0, 1, and bivalent variables are Boolean expressions.

(ii) if u and v are Boolean expressions, then so are $u \vee v$, $u \wedge v$, and $\bar{u}$

(iii) only items (i) and (ii) are Boolean expressions.

Branch-and-Bound (B & B) procedure: see section 3.2.1

Branching rules determine the order in which feasible subsets of solutions are partitioned, during the branch-and-bound procedure.

In a search procedure for the optimization of an SIDS, the candidate collection, is a superset of the optimal collection of sets. It is constructed as the first stage of the search procedure.

Given a set of objects, with pairwise similarity/dissimilarity measures defined, a classification method groups the objects into collections which are similar.

A Boolean expression A covers a Boolean expression B if for all evaluations of the variables of B for which B is true, A is true.

A data item is an undefined term, intuitively thought of as corresponding to a record in a file.

In a file system, descriptive storage is that information whose sole purpose is to facilitate access to basic storage.

In a dictionary-look-up file system every record has associated with it a unique identifier called a key. Each key is stored in the dictionary (a table), together with the

address of the record to which it applies. The address of a record corresponding to a key is determined by locating the appropriate key-address pair in the table.

Given a set of n Boolean expressions f_1, f_2, ..., f_n an <u>f-cover</u> of Boolean expression B is a Boolean expression $C(f_1,f_2,...,f_n)$ which covers B.

In the branch-and-bound algorithm, a <u>feasible subset</u> $\langle j,D,C,'P'\rangle$ of solutions is the collection of descriptive storage arrangements such that:

(i) for n candidates $\{S_1,S_2,...,S_n\}$, $j<n$; and

(ii) $D \subseteq \{S_1,S_2,...,S_j\}$; and

(iii) C is a lower bound for expected system cost when descriptive storage is $D \cup D'$ where $D' \subseteq \{S_{j+1},S_{j+2},...,S_n\}$; and

(iv) the algorithm has not yet detected a descriptive-storage arrangement with a cost less than C.

The <u>feed-back model</u> is a proposed model of a computerized file system whose main components are discussed in section 2.5.

<u>Functional covering:</u> same as f-cover

The <u>general retrieval algorithm</u> is an algorithm, proposed by Hsaio and Harary, which retrieves all records satisfying a query, in a generalized file structure.

A <u>generalized file structure</u> is a file together with a directory defined in a special manner (see section 4.2).

The generalized trace algorithm is an algorithm for retrieval using multilist files in which the lists are ordered according to their addresses.

The Idealized Descriptive Storage Model (IDSM) is a model of system cost for descriptive storage systems.

A Boolean expression A implies a Boolean expression B if and only if B covers A.

An inverted file is a generalized file structure in which every directory entry has the form $(K,n,n;a_1,a_2,\ldots,a_n)$; i.e., there are n records containing the keyword K and the addresses of these records are all found in the directory.

The Quine-McClusky Method is an algorithm to produce a minimum-literal union-of-intersections (intersections-of-unions) Boolean expression, equivalent to an arbitrary Boolean expression.

In a file system, manipulative cost is that cost which is attributable to retrieval, update, and other maintenance activity.

Given a set of Boolean expressions f_1, f_2, $\ldots$, f_n an f-cover C is a minimal f-cover of a Boolean expression B if and only if

(i) C is an f-cover of B; and

(ii) any other f-cover C' of B covers C.

Given a query q and an STDS $\langle I,A,D,\mathcal{S}\rangle$, the retrieval set with respect to q is the collection $\{d|(d\in I)(E(q,d)=1)\}$ of all data items for which q is true.

When applied to the optimization of STDS, a search procedure is a method by which

(i) a collection of candidates, for inclusion in the optimal collection of descriptive storage, is constructed

(ii) members of the optimal collection are chosen from this collection of candidates.

In an STDS $\langle I,A,D,\mathcal{S}\rangle$, the set contents of a set $S\in\mathcal{S}$, are those data items in the set $\{d|(d\in I)(E(D(S),d)=1)\}$.

In an STDS $\langle I,A,D,\mathcal{S}\rangle$, the set definition of a set $S\in\mathcal{S}$, is a Boolean expression D(S) which determines the contents of S.

A Set-theoretical Descriptive Storage (STDS) is a four-tuple $\langle I,A,D,\mathcal{S}\rangle$ where

(i) I is a finite set of data items (records), also called basic storage.

(ii) A is a finite set of Boolean conditions (attributes), each of which may be evaluated as either true or false for every data item $d\in I$.

(iii) $\Delta:\beta(A)\rightarrow\top\top(I)$ defines a set for every Boolean expression e of $\beta(A)$ as follows:

$$\Delta(e) = \{d|(d\in I)(E(d,e)=1)\}$$

(iv) $\mathcal{S} \subseteq \{\Delta(e)|e\in\beta(A)\}$. As a notational convenience, we write D(S) for the expression which generated the

labelled set SE$.

The storage cost of a file system, for a period of time, is that cost charged for the storage of basic and descriptive storage.

A set X is a superset of a set Y if all members of Y are also members of X.

Tangible cost refers to cost or cost elements of resources which can be conventionally measured by, for example, computer accounting algorithms. These costs are to be distinguished from intangible costs which cannot be conventionally measured, i.e., the cost to time-sharing users of a computer system attributable to poor response time.

As applied to file systems, an update occurs when any part of the file system is changed. The changes include the addition or destruction of data items and the changing of the attributes possessed by data items.

A Wong-and-Chiang Descriptive Storage system is an STDS system $\langle I, A, D, S\rangle$ where descriptive storage is generated by a set $A' \subseteq A$ to consist of all non-null subsets of I with a definition $D(S) = \bigwedge_{a \in A} \hat{a}$ where $\hat{a}$ is a or $\bar{a}$.

Work-set Systems is a proposed class of STDS systems in which decisions regarding the constitution of descriptive storage are made analogously to the manner in which replacement algorithms operate in paging systems.

APPENDIX B: Notation

This appendix gives the notation used in several sections of the thesis. It may be used as a reference when reading these sections.

Section 1.4

An STDS is a four-tuple $\langle I, A, D, \mathcal{S} \rangle$ where

I : basic storage (i.e., the data file)

A : a collection of attributes

$\mathcal{S}$: a collection of sets describing I (descriptive storage)

D(S) : the definition of a set $S \in \mathcal{S}$

Section 3.1.2

While developing a formula specifying total cost of an STDS system, we use:

S_j : bivalent variable specifying whether or not the i-th candidate is defined.

T_j : a term corresponding to the j-th arrangement

C_j : the cost of the j-th arrangement

Section 3.2.1

$\{S_1, S_2, \ldots, S_n\}$ is a candidate collection.

A subset of the valuations of the variables used in a B&B optimization is denoted $\langle j, D, Cmin, X \rangle$

j : number of candidates considered in subset.

D : of the j candidates considered, those which are defined

Cmin : a lower bound on cost for this subset.

X : a symbol indicating whether the subset is feasible

The following symbols are used while demonstrating that partitioning determines an optimal collection of candidates.

S_i : candidate collection for the i-th partition

Q_i : query collection for the i-th partition

f(q): frequency of the query q.

R(q,X) : cost of performing retrieval when the descriptive storage is the collection X.

X : any collection of candidates in the i-th partition.

Section 4.2

A directory entry in a generalized file structure has the form:

$(K,n,h;a_1,a_2,\ldots,a_h)$

K : keyword

n : number of records for which K is true

h : number of K-lists for keyword K

a_i : $(1 \leq i \leq h)$ address of a record at the start of a K-list

C.1-5:lemma: If $f ==> g$, then $\bar{g} ==> \bar{f}$.

C.1-6:lemma: $f ==> g_1 \wedge g_2$ if and only if $f ==> g_1$ and $f ==> g_2$.

C.1-7:lemma: $f_1 \vee f_2 ==> g$ if and only if $f_1 ==> g$ and $f_2 ==> g$.

As all the above lemmas may be proven similarily, we will only exhibit the proof for C.1-6.

proof of C.1-6

(sufficiency). Assume $f ==> g_1$ and $f ==> g_2$. Then, by the definition of covering, for any evaluation of the variables such that f is true, it follows that g_1 is true and that g_2 is true. Thus, for any such valuation $g_1 \wedge g_2$ is true and so $f ==> g_1 \wedge g_2$.

(neccessity). Assume $f ==> g_1 \wedge g_2$. Then, whenever f is true, $g_1 \wedge g_2$ is true. Hence, whenever f is true, g_1 is true and g_2 is true. Thus, $f ==> g_1$ and $f ==> g_2$.

This completes the proof.

These lemmas will be referred to in section C-2 when we specify the construction of minimal f-covers. An example of this construction is given in section C-3. In section C-4 we will introduce a number of restrictions on the form of f-covers which we permit. We will indicate how to construct these f-covers. Several remarks on how to simplify the expressions for f-covers will be made in section C-5. Before concluding our discussion in section C-7 we develop several

(ii) Each term B_i is in the form $B_i = \bigwedge_{j=1}^{k} \tilde{v}_j$ where $\tilde{v}_j$ stands for the j-th variable of V or its complement. For every term B_i, we define a valuation of the variables in V as follows: v_j is true if $\tilde{v}_j$ is not complemented in B_i and v_j is false otherwise.

(iii) Each f_j for j=1,2, ..., n is either true or false for the valuation of V corresponding to B_i. Thus, for i=1, 2, ..., l, we define a term $C_i = \bigwedge_{j=1}^{n} \tilde{f}_j^i$ where $\tilde{f}_j^i = f_j$ if f_j is true for this valuation and $\tilde{f}_j^i = \bar{f}_j$ otherwise.

(iv) We define $C(B) = \bigvee_{i=1}^{l} [\bigwedge_{j=1}^{n} \tilde{f}_j^i]$.

We claim that C(B) is a minimal f-cover of B. We shall first demonstrate that C(B) is an f-cover of B.

C.2-4: lemma: C(B) is an f-cover of B.

proof:

Consider any term B_i in the disjunctive normal form of B. The valuation defined in C.2-3(ii) results in B_i evaluating as true. Any other valuation results in B_i evaluating as false. By the definition in C.2-3(iii) it follows that $\tilde{f}_j^i$ is true whenever B_i is true. Hence, we have $B_i \Longrightarrow \tilde{f}_j^i$ for $1 \le i \le l$ and $1 \le j \le n$. By repeated application of lemma C.1-6 we thus derive $B_i \Longrightarrow \bigwedge_{j=1}^{n} \tilde{f}_j^i$. Repeated application of lemma C.1-7 proves that $\bigvee_{i=1}^{l} B_i \Longrightarrow \bigvee_{i=1}^{e} [\bigwedge_{j=1}^{n} \tilde{f}_j^i]$. In other words, $B \Longrightarrow C(B)$ and so C(B) is an f-cover.

We will now show that C(B) is minimal.

C.2-5: Lemma: C(B) is a minimal f-cover.

proof:

Let $C'(f_1,f_2,\ldots,f_n)$ be any f-cover of B. Write $C' = \bigvee_{i=1}^{l'} C'_i$ in disjunctive normal form with $f_1, f_2, \ldots, f_n$ as variables. For any term C_i, $1 \le i \le l$, defined in the construction of C(B) we shall show that there exists an integer t, $1 \le t \le l'$, such that $C_i = C'_t$.

Consider any term C'_m, $1 \le m \le l'$. We can write $C'_m = \bigwedge_{j=1}^{n} \tilde{g}_j^m$ where $\tilde{g}_j^m$ is f_j or its complement. Comparing C'_m to C_j, suppose that there exists a j such that $\tilde{f}_j^i$ is not identical to $\tilde{g}_j^m$. Then, $\tilde{g}_j^m$ is the complement of $\tilde{f}_j^i$. Now, for the evaluation defined by B_i and defining C_i, $\tilde{f}_j^i$ is true and so $\tilde{g}_j^m$ is false. Hence, C'_m is false. Hence, all terms not identical to C_i are false. But, for the evaluation in question, P is true and so C' is true. Thus, there must exist a t, $1 \le t \le l'$, such that $C_i = C'_m$.

Hence, we may write $C' = C(B) \vee C''$ and so C' is true whenever C(B) is true, i.e., $C(B) \Rightarrow C'$ which proves the minimality of C(B).

To illustrate the method of constructing an f-cover, we present an example in section C.3.

Before considering the example, we remark that all minimal f-covers are equivalent to one another. For two minimal f-covers, C_1 and C_2, we have $C_1 \Rightarrow C_2$ and $C_2 \Rightarrow C_1$. We can thus show that C_1 and C_2 have the same value for any valuations of the variables in V.

C-3 EXAMPLE OF CONSTRUCTION OF MINIMAL F-COVER

This section is intended to illustrate the construction of minimal f-covers as proposed in section C.2. Let B, f_1, f_2, and f_3 be defined as:

$B = (a \vee b) \wedge d$

$f_1 = a$

$f_2 = b$

$f_3 = a \wedge c$

Hence, the set of variables is $V = \{a,b,c,d\}$. The disjunctive normal form of B is given by

C.3-1 $B = abcd \vee ab\bar{c}d \vee a\bar{b}cd \vee a\bar{b}\bar{c}d \vee \bar{a}bcd \vee \bar{a}b\bar{c}d$

We have omitted '$\wedge$' operators to increase legibility. Steps (ii) and (iii) of the construction are summarized in table C.3-2 below.

C.3-2 Summary of Steps (i) and (iii)

	Evaluation of V	
B_I	a b c d	C_I
$abcd$	1 1 1 1	$f_1 f_2 f_3$
$ab\bar{c}d$	1 1 0 1	$f_1 f_2 \bar{f}_3$
$a\bar{b}cd$	1 0 1 1	$f_1 \bar{f}_2 f_3$
$a\bar{b}\bar{c}d$	1 0 0 1	$f_1 \bar{f}_2 \bar{f}_3$
$\bar{a}bcd$	0 1 1 1	$\bar{f}_1 f_2 \bar{f}_3$
$\bar{a}b\bar{c}d$	0 1 0 1	$\bar{f}_1 f_2 \bar{f}_3$

Hence, we derive

C.3-3 $C(B) = f_1 f_2 f_3 \vee f_1 f_2 \bar{f}_3 \vee f_1 \bar{f}_2 f_3 \vee f_1 \bar{f}_2 \bar{f}_3 \vee \bar{f}_1 f_2 \bar{f}_3$

$= f_1 \vee f_2 \bar{f}_3$

as a minimal f-cover.

The generality of the method is illustrated by the fact that a non-trivial f-cover was constructed even though a variable 'd' was present only in B. We note that C(B) can be written as

C.3-4 $C(B) = (f_1 \vee f_2)(f_1 \vee \bar{f}_3)$

and by substitution for the expressions f_1, f_2, f_3 we derive

C.3-5 $f_1 \vee f_2 = a \vee b$

C.3-6 $f_1 \vee \bar{f}_3 = a \vee \overline{(\bar{a}c)}$

$= 1$

and so

C.3-7 $C(B) = a \vee b = (f_1 \vee f_2)$

Simplifications of the above nature are discussed in section C-5. As the remarks apply to f-covers in general we will consider first a number of other f-covers in the following section.

C.4 RESTRICTED F-COVERS

As indicated in section 2.2 of the thesis, minimal f-covers may not be the preferred form of f-cover to be derived. For example it may be desireable to obtain f-covers in which no complementation occurs. Hence, we introduce a number of restrictions on the composition of f-covers. For each restriction we indicate a construction procedure and show that such f-covers exist, then the appropriate method will derive them.

We propose that f-covers be restricted in any of the following ways.

C.4-1 (i) No complementation be allowed in the expression for the f-cover.

(ii) No '$\vee$' operators be allowed in the f-cover expression.

(iii) No '$\wedge$' operators be allowed in the f-cover expression.

or (iv) The expression for the f-cover is to have property (i) and either property (ii) or (iii).

Property (ii), for example, specifies that the f-cover must have the form:

C.4-2 $C = \bigwedge_{i \in Q} \mathbf{f}_i$ where $Q \subseteq \{1,2,\ldots,n\}$ and $\mathbf{f}_i$ is f_i or its complement

To construct an f-cover with property (iii) of C.4-1 we proceed as follows:

C.4-3

(i) For the given Boolean expression B construct a minimal f-cover using the method specified in C.2.

(ii) Write the resultant f-cover as an 'AND' -of- 'OR' terms expression.

(iii) Select one of the 'OR'-terms as the required f-cover.

By lemma C.1-6 and the definition of an f-cover, the selected term is an f-cover in the required format. The problem of which term to select is based upon other considerations and is discussed in the thesis. Referring to the example in section C.3, we derived a minimum f-cover C.3-3 which was written in the form specified in C.4-3 (ii). From C.3-4 we see that either of the terms $(f_1 \vee f_2)$ or $(f_1 \vee \bar{f}_3)$ could have been selected. We observe that step (ii) of C.4-3 is always possible since C(B) can always be written in conjunctive normal form.

We now consider the situation where f-covers are restricted according to C.4-1 (ii), i.e., f-covers are of the form $U = \bigwedge_{i \in Q} \tilde{f}_i$ where $Q \subseteq \{1,2,\ldots,n\}$. Suppose that such an expression exists for an expression B. Then, by lemma C.1-6 and the definition of an f-cover, it follows that $\tilde{f}_i$, $i \in Q$,, is an f-cover of B. Writing $B = \bigvee_{j=1}^{l} B_j$ in disjunctive normal form,

$\bar{f}_i$ is an f-cover of B_j, $j \in \{1,2,\ldots,l\}$, by lemma C.1-7. In other words, we can construct the required f-cover as the 'AND' of all f_i or their complements which occur in the same form in all terms of the disjunctive normal form of the minimal f-cover. As an example let

C.4-4 $B = a \wedge b$

$f_1 = a \vee c$

$f_2 = \bar{b} \wedge \bar{c}$

$f_3 = c$

$f_4 = \bar{c}$

Then, an appropriate f-cover is $(f_1 \wedge \bar{f}_2)$ since, for the two terms in the disjunctive normal form of the minimal f-cover, f_1 occurs in uncomplemented form and f_2 occurs in complemented form.

For the case where we want an f-cover which involves no complementation we adapt the three procedures for f-cover construction. When the only restriction is that complementation is not allowed we modify the method for constructing minimal f-covers as follows. Instead of creating the disjunctive normal form of the minimal f-cover, we create terms which omit any complemented forms of $f_1, f_2, \ldots, f_n$. Referring to the example in section C.3, we create by this method a table corresponding to table C.3-2.

C.4-5 Table corresponding to table C.3-2

Evaluation of V

B_i	a b c d	C_i
$abcd$	1 1 1 1	$f_1 f_2 f_3$
$ab\bar{c}d$	1 1 0 1	$f_1 f_2$
$a\bar{b}cd$	1 0 1 1	$f_1 f_3$
$a\bar{b}\bar{c}d$	1 0 0 1	f_1
$\bar{a}bcd$	0 1 1 1	f_2
$\bar{a}b\bar{c}d$	0 1 0 1	f_2

and so we derive

C.4-6 $C = f_1 f_2 f_3 \vee f_1 f_2 \vee f_1 f_3 \vee f_1 \vee f_2 \vee f_2$

$= f_1 \vee f_2$

as an f-cover for $B = (a \wedge b) \vee d$. As long as every term in the disjunctive normal form of B is covered, the outlined procedure is valid, as could be proven similarily to lemma C.2-4. In the case where the disjunctive normal form of the minimal f-cover contains a term $\bar{f}_1 \wedge \bar{f}_2 \wedge \ldots \wedge \bar{f}_n$, we claim that the only f-cover which does not involve complementation is the trivial f-cover 1, i.e., the expression which is true for all valuations. This is a consequence of the following lemma.

C.4-7 lemma Let E be a Boolean expression which does not involve complementation and let the variables occuring in E be $v_1, v_2, \ldots, v_n$. Then, the

disjunctive normal form of E contains the term $\bar{v}_1 \wedge \bar{v}_2 \wedge \cdots \wedge \bar{v}_n$ if and only if E = 1.

Informally, the proof is outlined as follows. Sufficiency is trivial and necessity can be proven using induction on n, the number of variables. The basis step (n=1) is established by considering the four possible disjunctive normal forms which E may have. The induction step proceeds as follows.

C.4-8 (i) Factor F into terms involving v_1 and $\bar{v}_1$ using Shannon's expansion theorem.

(ii) By the induction hypothesis, the terms involving $\bar{v}_1$ reduce to $\bar{v}_1$.

(iii) Hence, E reduces to the form $F = \bar{v}_1 \vee v_1 F_1$ where F_1 does not involve the variable v_1.

(iv) Since E can be written without complementation, E_1 must be identical to 1 since this is the only way v_1 can be subsumed. Hence, F=1.

Lemma C.4-7, together with the result from the proof of lemma C.2-5 that any f-cover C can be written in the form $C = C(B) \vee C'$ where C(B) is a minimal f-cover, can be used to prove lemma C.4-9.

C.4-9 lemma. If the disjunctive normal form of a minimal f-cover for an expression B contains a term where all the given expressions are complemented, the

C.5 SIMPLIFICATION OF F-COVERS

The method for constructing minimal f-covers produces f-covers in disjunctive normal form. In the examples we manipulated these expressions into simpler forms. A systematic for this type of manipulation is the Quine-McClusky method [WOOD68,KOH70] which produces minimum-literal two-level Boolean expressions. More simply, the method produces either 'AND'-of-'OR'-terms or 'OR'-of-'AND'-terms expressions in which the number of times variables are used is minimum. An alternative but equivalent method has been proposed by Zissos and Duncan [ZIS73]. In both cases, the expressions produced are equivalent to the original expressions.

Further simplification of Boolean expressions may be accomplished by functional decomposition. In general, this process involves expressing a Boolean function of n variables as a composition of a number of functions, each depending on less than n variables. A general introduction and references to particular methods is found in Kohavi [KOH70].

The simplifications above involve rewriting f-covers in an equivalent manner, treating f_1, f_2, ..., f_n as variables. We have made no use of the fact that f_1, f_2, ..., f_n are themselves expressions depending on a (possibly) shared set of variables. It is often possible to eliminate terms in f-covers by using this information.

In general, we can write an f-cover C as an 'OR'-of'-'AND'-terms expression. In this case we can eliminate any term which is covered by another term. We determine whether

C.6 APPLICATION OF F-COVERS TO STDS

In this section we shall be concerned with the use of f-covers with regard to Set-theoretical Descriptive Storage (STDS). As a preliminary, we define:

C.6-1 Definition: An STDS is a four-tuple $\langle I, A, D, \mathcal{S} \rangle$ where

(i) I is a finite set of undefined elements called data items.

(ii) A is a set of Boolean conditions (attributes) such that for each $a \in A$ and each $d \in I$ a value, true or false, is defined and denoted $E(d,a)$. Similarly, for a Boolean expression B, depending on variables from the set A, $E(d,B)$ is the evaluation of expression B when all the variables of A are replaced by their values with respect to d.

Thus, each $S \in \mathcal{S}$ consists of all data items in I for which the definition D(S) of S is true.

(iii) $\Delta : \beta(A) \rightarrow \Pi(I)$ defines a set for every Boolean expression e of $\beta(A)$ as follows:

$$\Delta(e) = \{d \mid (d \in I)(E(d,e)=1)\} .$$

(iv) $\mathcal{S} \subseteq \{ \Delta(e) \mid e \in \beta(A) \}$. As a notational convenience, we write D(S) for the expression which generated the labelled set $S \in \mathcal{S}$.

that $F(d,D(S_1))=F(d,D(S_2))=1$. Hence, by definition of STDS, $d \in S_1$ and $d \in S_2$. Hence, $d \in S_1 \cap S_2$ and so $S_1 \cap S_2 \subseteq \{d \mid (d \in I)(E(d,D(S_1) \wedge D(S_2))=1)\}$.

This completes the proof of (i).

Thus, we may construct a set of data items, corresponding to an f-cover of an expression B, by performing the set-theoretical operations in an expression defined as follows. Take the expression for the f-cover and perform the following (simultaneous) substitutions.

C.6-3 (i) Replace '$\wedge$' by '$\cap$'.

(ii) Replace '$\vee$' by '$\cup$'.

(iii) For all $S \in \mathcal{S}$ replace $D(S)$ by S.

Thus, the expression $C = (D(S_1) \vee D(S_2)) \wedge D(S_3)$ becomes an expression $(S_1 \vee S_2) \wedge S_3$. It is now apparent why the simplifications in section C.5 are important: the set corresponding to the f-cover in question can be constructed in a correspondingly easier manner.

For an expression B, the minimal f-cover will have the least number of data items. This is shown by the following lemma and its corollary.

C.6-4 lemma: In the STDS $\langle I,C,D,\mathcal{S} \rangle$ where $S_1, S_2 \in \mathcal{S}$, if $D(S_1) \Longrightarrow D(S_2)$, then $S_1 \subseteq S_2$.

C.6-5 corollary: In the STDS $\langle I,C,D,\mathcal{S} \rangle$, let B be an

expression depending on variables in C. Then the number of data items in the set corresponding to a minimal f-cover of B is minimum.

Lemma C.6-4 may proven similarily to lemma C.6-2. The corollary follows from lemma C.6-4 and the definition of a minimal f-cover.

C.7 CONCLUSIONS

In conclusion, we have developed a theory whereby covers may be constructed for arbitrary Boolean expressions. These covers, called f-covers, are expressions where other Boolean expressions are treated as variables.

Various constructions were presented and justified. The basic construction, introduced in section C.2, specified how to obtain a special f-cover called a minimal f-cover. The methods in section C.4 defined f-covers with other restrictions placed upon them. In section C.6 we developed some results for f-covers used in conjunction with Set-theoretical Descriptive storage. We were concerned with construction of sets of data items which corresponded to f-covers. Thus, the simplifications introduced in section C.5 were indicated to be useful.

The purpose of the appendix has been to develop a formal theory of f-covers. Reference to the results obtained in this appendix are made in the thesis. As is indicated, this theory may be used to define methods for retrieval of information in STDS file systems.

Appendix D: Case study: GENASYS

This appendix is a case study of a file currently used at the University of Waterloo to provide mailing labels for the university community. The system, named GENASYS (GENeral Addressing SYStem), is presently implemented as a sequential file stored on magnetic tape. The study is intended to investigate the implementation of this system as an inverted file stored on an IBM 2314 direct-access storage device (a disk). We shall choose the optimal collection of inverted lists, based upon how the system was used during a period of 35 working days.

Each record in the GENASYS file corresponds to an individual. These records consist of fields, each of which contains an item of information. The data in these fields includes:

D-1: (i) name of the individual
(ii) title of the individual
(iii) home address of the individual
(iv) business address of the individual
(v) a collection of mailing-list names
(vi) the GENASYS number of the individual

The GENASYS number is an eight-character field which uniquely identifies each individual (the GENASYS number of the author is "WEL06500"). Various people and institutions within the university have specified collections of people to be included

on mailing lists. The lists are identified by three-character names ("X08" is the name of the mailing list for the University of Waterloo Computer Centre Newsletter). A record contains the names of those lists of which the individual is a member. The relevant statistics concerning the GENASYS file are summarized in table D-2.

Requests for mailing labels are initiated by various persons and institutions within the university. A special form is used to specify

D-3

(i) the format of the mailing label

(ii) the order in which the labels are printed

(iii) the criteria to be used to select records from which the labels are to be constructed.

A department within the university manually ensures that unauthorized access to the mailing lists is prevented. The requests are collected, encoded, and submitted in a batch to a program which produces the required labels.

A collection of mailing lists is specified on each request form. The program produces a label for every person that occurs on any of these lists. Thus, only the restrictive query format

D-4

$$q = L_1 \vee L_2 \vee \cdots \vee L_k$$

is allowed, where L_i is treated as an attribute with the meaning "person is a member of mailing list L_i".

We shall investigate the implementation of GENASYS using

the direct-access storage at the University of Waterloo Computing Centre. The system shall be modelled as an inverted file and we shall use the branch-and-bound (B&B) method to determine the optimal collection of inverted lists. With the cooperation of the Data Processing Department at the University of Waterloo, all requests for mailing labels were collected for a period of 35 working days. 135 requests were collected and will be used as a characterization of how the system is expected to be used.

In the proposed system, the inverted lists will be assumed to have the following characteristics:

D-5 (i) Every data item (record) in an inverted list is represented as a pointer which is to be stored in a format requiring four characters.

(ii) These pointers are stored in blocks of b pointers each. An access to direct-access memory is required to read a block of pointers into the memory of the computer.

(iii) There is no ordering of the pointers within an inverted list or within a block.

Because of the restrictive format D-4 of the queries, two methods of determining the retrieval set for a query will be used:

D-6: (i) If inverted lists exist for all of the k attributes in query D-4, then the retrieval algorithm D-7 will be used.

(ii) If the condition in (i) is not true for a given query, then the retrieval set for q will be determined by reading every record in the file and choosing those which satisfy q.

In the case where all the k attributes in query D-4 have inverted lists to correspond to them, we determine the retrieval set for q as follows:

D-7: (i) Dispersion phase: For each of the blocks of pointers in the list for L_i ($1 \leq i \leq k$), order the pointers within a block.

(ii) Merge phase: Use a balanced two-way merge to order the pointers.

(iii) Elimination phase: Retrieve the records identified by the sorted pointers and evaluate the query for each record. When the evaluation is true, add the record in question to the retrieval set (i.e., produce a label for the record).

Let there be b(i) blocks of pointers for the list L_i ($1 \leq i \leq k$) where

D-8
$$n = \sum_{i=1}^{k} b(i)$$

and suppose that there are m unique pointers at the end of the merge phase ($m \leq n$).

We shall assume that the accounting procedures are

(constant) cost to evaluate q for a record.

(iv) We shall assume that a (constant) cost C_0 is required to determine whether or not the algorithm D-7 may be used in preference to reading all records.

Combining the factors in D-10, we derive

D-11: $$\text{cost} = C_0 + (D+2\cdot A)n + p(n\cdot M+2\cdot n\cdot A) + (m+n)\cdot A + m\cdot E$$
$$= C_0 + n\cdot D + p\cdot n\cdot M + m\cdot E + A\cdot\{2\cdot n + 2\cdot n\cdot p + m + n\}$$

When one or more of the attributes in query D-4 does not have an inverted list defined, the cost of sequentially reading the file to determine a retrieval set is assumed to be a (constant) S.

The determination of the optimal collection of inverted lists was accomplished in three phases:

D-12: (i) A program was written to perform the retrievals for the queries, according to algorithm D-9. This program produced an accounting record (D-13) for every retrieval performed.

(ii) The queries were partitioned according to the equivalence relation θ (D-15).

(iii) For every partition, the optimal sub-collection of candidates was determined using the branch-and-bound method outlined in chapter three.

The first stage simulated the retrievals using an extract of

the actual GENASYS file, when all 87 lists are defined. The accounting record for every query contains the following information:

D-13: (i) the query

(ii) CPU time spent scanning, to decide if algorithm D-7 could be used.

(iii) CPU time for dispersion phase

(iv) CPU time for merge phase

(v) CPU time for elimination phase

(vi) the number of blocks of pointers

(vii) the number of passes in the merge phase

(viii) the number records read in elimination phase

(ix) the number records in the retrieval set

(x) the number of accesses to DASD

From this information the constants C_0, D, M, and F were estimated. The constant S was estimated by the relationship:

D-14: $S = C_0 + r \cdot F + r \cdot A$

where r is the number of records in the file. We define an equivalence relation θ in the manner described in 3.2.1 as follows:

D-15: (i) For a query $q = L_1 \vee L_2 \vee \cdots \vee L_k$, define $K(q) = \{L_1, L_2, \cdots, L_k\}$.

(ii) For any two queries q_1, q_n, $q_1 \theta q_n$ if and only if there exists a set of queries $q_1, q_2, \cdots, q_n$ such that $K(q_i) \cap K(q_{i+1}) \neq \emptyset$.

A program was written to read the accounting file D-13 and to

partition the queries into equivalence classes of the relation θ. The 135 queries were divided into 35 groups. Recalling the analysis of the number of steps in the branch algorithm,

D-16: $135 \cdot 2^{87} = 2.089 \cdot 10^{28}$ (approximately)

is the maximum number of steps when partitioning is not used. The maximum number of steps when partitioning was used is 11,520. The actual number of steps was 310 and 17 (out of a possible 87) lists were in the optimal collection. The following CPU times (IBM 360/75 computer) were required in the three stages:

	PHASE	CPU TIME (in minutes)
D-17	1	.35
	2	.07
	3	.16

All programs were written in PL/I, augmented by several subroutines written in 360 Assembly Language.

APPENDIX E: Derivation of conditional probabilities for transition in multilist file

In this appendix we derive the probability of a transition from a record with the attributes

E-1: $\{A_1, A_2, \ldots, A_k, A_{k+1}, \ldots, A_s\}$

to a record with attributes

E-2: $\{A_1, A_2, \ldots, A_k, A_{s+1}, \ldots A_{s+u}\}$

when the query being processed by the generalized trace algorithm is of the form

E-3: $q = A_1 \wedge A_2 \wedge \ldots \wedge A_t$

We assume that the probability that an arbitrary record has an attribute A_i $(1 \leq i \leq t)$ is given by p_i and that the attributes are distributed throughout the records of the file uniformly and independently of one another.

Let us suppose that a transition from a record with the attributes E-1 to a record with the attributes E-2 occurs n records after the original record. This transition can occur if and only if all the following conditions are true:

E-4:
(i) A record exactly n records following the original record has exactly the attributes E-2;
(ii) At least one of the attributes A_1, A_2, ..., A_k does not occur in the intervening n-1 records.
(iii) All of the attributes A_{k+1}, A_{k+2}, ..., A_s occur in the intervening (n-1) records.

The probability P_1 that condition(i) of E-4 is true is given by:

E-5: $$P_1 = \left[\prod_{i=1}^{k} p_i\right]\left[\prod_{i=k+1}^{s} (1-p_i)\right]\left[\prod_{i=s+1}^{s+u} p_i\right]\left[\prod_{i=s+u+1}^{t} (1-p_i)\right]$$

under the assumptions above.

The probabilities that conditions (ii) and (iii) of E-4 are true are derived as follows. The probability that an arbitrary record does not have attribute A_i $(1 \le i \le t)$ is given by $(1-p_i)$. The probability that n-1 intervening records do not have this attribute is given by $(1-p_i)^{n-1}$. Hence, the probability $P_2(n)$ that condition (ii) of E-4 is true, and the probability $P_3(n)$ that condition (iii) of E-4 is true are given by:

E-6: $$P_2(n) = 1 - \prod_{i=1}^{k} (1-(1-p_i)^{n-1})$$

$$P_3(n) = \prod_{i=k+1}^{s} (1-(1-p_i)^{n-1})$$

Then the probability of transition is given by

E-8: $$P = \sum_{n=1}^{\infty} P_1 \cdot P_2(n) \cdot P_3(n)$$

$$= P_1 \cdot \sum_{n=1}^{\infty} \left[1 - \prod_{i=1}^{k} (1-(1-p_i)^{n-1})\right]\left[\prod_{i=k+1}^{s} (1-(1-p_i)^{n-1})\right]$$

$$= P_1 \cdot \sum_{n=1}^{\infty} \left[\prod_{i=k+1}^{s}(1-(1-p_i)^{n}) - \prod_{i=1}^{s} (1-(1-p_i)^{n})\right]$$

In order to find the limit of the series E-8, we shall first find some identities.

As we shall show, the series

which is the required limit.

As an example, we shall use small letters to represent the probabilities in the sets

E-15: $H_1 = \{1-a, 1-b\}$

$H_2 = \{1-a\}$

when the complete set of attributes is {a,b,c,d}. Then, we have

E-16: $P_1 = a \cdot (1-b) \cdot c \cdot (1-d)$

$H_1 = \{1-a, 1-b\}$

$H_2 = \{1-a\}$

$$L(H_1) = \frac{1}{a} + \frac{1}{b} - \frac{1}{1 - (1-a)\cdot(1-b)}$$

$$L(H_2) = \frac{1}{a}$$

$$P = a \cdot (1-b) \cdot c \cdot (1-d) \cdot \left\{ \frac{1}{b} - \frac{1}{1 - (1-a)\cdot(1-b)} \right\}$$

representing the probability of a transition from a record with the attributes {a,b} to a record with the attributes {a,c}.

Appendix F: Bibliography

ABR70: "Data base structure (design and implementation)", J.R.Abrial, G.Beaume, G.Henneron, R.Morin, G.Vigliano, International Summer School on Information Systems-techniques and Implementation, Copenhagen, (Aug. 1970)

ARON69: "Information systems in perspective", J. D. Aron, Computing Surveys, Vol. 1, No. 4, (Dec. 1969)

ARORA69: "Randomized binary search technique", S. R. Arora, CACM, Vol. 12, No. 2, (Feb. 1969)

ASPIN72: "Data base re-organization - concepts", L. Aspinall, C. J. Bell, T. W. Rogers, IBM Tecnical Report UKSC-0011, (Feb. 1972)

ATW72: "Effects of secondary storage I/O contention on the performance of an interactive information management system", R.C.Atwood, Proc. ACM 1972

BACH72: "The evolution of storage structures", C. W. Bachman, CACM, Vol. 15, No. 7, (July 1972)

BACH73: "The programmer as a navigator", C. W. Bachman, CACM, Vol. 16, No. 11, (Nov. 1973)

BAYE70: "Organization and maintenance of large ordered indices", R. Bayer, E. McCreight, SIGFIDET 1970

BAYE71: "Binary B-trees for virtual memory", R. Bayer, SIGFIDET 1971

BELZ64: "Theoretical considerations in information retrieval systems", J. Belzer, W. Coffman, CACM, Vol. 7, No. 7, (July 1964)

BIG73: "Specialized languages: an applications methodology", R. H. Bigelow, N. R. Greenfield, P. Szolovits, F. B. Thompson, Proc. of 1973 National Computer Conference and Exposition, (June 1973)

BLOOM70: "Space/time trade-offs in hash coding with allowable errors", B. H. Bloom, CACM, Vol. 13, No. 7, (July 1970)

BONN64: "On some clustering techniques", R. E. Bonner, IBM Journal of research and development, Jan. 1964

BRAC72: "A relational data base management system", G. Bracchi, A. Fedeli, P. Paolini, Proc. ACM 1972

BRENT73: "Reducing the retrieval time of scatter storage techniques", R. P. Brent, CACM, Vol. 16, No. 2, (Feb. 1973)

BYRON73: "Representation of sets on mass storage devices for information retrieval systems", S. T. Byron, W. T. Hardgrave, Proc. of 1973 National Computer Conference and Exposition (June 1973)

CARD73: "Evaluation and selection of file organization - a model and system", A. F. Cardenas, CACM, Vol. 16, No. 9, (Sept. 1973)

CAUS65: "Some mathematical problems arising in information retrieval from inverted files", R. L. Causey, Redstone Scientific Information Centre, Report 423, (1965) (distributed by NTIS: AD-619-955)

CHANG72: "Dynamic programming as applied to feature subset analysis in a pattern recognition system", C. Y. Chang, Proc. ACM 1972

CHILD68A: "Feasibility of a set-theoretic data structure", D. L. Childs, Proc. IFIPS 1968

CHILD68B: "Description of a set-theoretic data structure", D. L. Childs, Technical Report 3, Concomp Project, University of Michigan (1968)

CHU69: "Optimal file structure in a multiple computer system", W. W. Chu, IEEE Transactions on Computers, vol. C-18, no. 10, (Oct. 1969)

CLAMP64: "Randomized binary searching with tree structures", H. A. Clampett, CACM, Vol. 7, No. 3, (Mar. 1964)

CODAS69A: "Data Base Task Group Report", CODASYL Programming Languages Committee, (Oct. 1969 revised June 1971)

CODAS69B: "A survey of data base management systems", CODASYL Systems Committee, (May 1969)

CODAS71: "Feature analysis of generalized data base management systems", CODASYL Systems Committee, (April 1971)

CODD70: "A relational model of data for large shared data banks", E. F. Codd, CACM, Vol. 13, No. 6, (June 1970)

CODD71: "A relational data base sublanguage founded on the relational calculus", E.F.Codd, SIGFIDET 1971

COFF70: "File structures using hashing functions", E. G. Coffman, J. Eve, CACM, Vol. 13, No. 7, (July 1970)

COUL72: "Towards content-addressing in data bases", G.F.Coulouris, J.M.Evans, R.W.Mitchell, Computer Journal, Vol.15, No. 2, (May 1972)

CUNN72: "Evaluation of hierarchical grouping techniques: a preliminary study", K. M. Cunningham, J. C. Ogilvie, Computer Journal, Vol. 15, No. 3, (Aug. 1972)

DATE71A: "File definition and logical data independence", C.J.Date, P.Hopewell, SIGFIDET 1971

DATE71B: "Storage structure and physical data independence", C.J.Date, P.Hopewell, SIGFIDET 1971

DAY65: "On optimal extracting from a multi-file data storage system: an application of linear programming", R. H. Day, Operations Research, Vol. 13, No. 3, (May 1965)

DEM71: "Storage optimization of tree structured files",P.A.D. deMaine, T.Rotwitt Jr., SIGFIDET 1971

DODD69: "Elements of data management systems", G. G. Dodd Computing Surveys, Vol. 1, No. 2, (June 1969)

DZUB63: "File organization for automated retrieval (a description)", B. Z. Dzubak, C. R. Warburton, IBM technical report TR 00.946 (Aug. 1963)

EARL71: "Toward an understanding of data structures", J. Earley, CACM, Vol. 14, No. 10, (Oct. 1971)

FELL50: "An introduction to probability theory and its applications", W.Feller, John Wiley & Sons, (1950)

FLOR71: "Average binary search length for dense ordered lists", I. Flores, G. Madpis, CACM, Vol. 14, No. 9, (Sept. 1971)

FOST73: "A generalization of AVL trees", C. C. Foster, CACM, Vol. 16, No. 8, (Aug. 1973)

GAUT70: "Designing system programs", R. L. Gauthier, S. D. Ponto, Prentice-Hall Inc. (1970)

GREEN72: "Computer system support for data analysis", Norton Greenfield, Ph. D. dissertation, California Institute of Technology, (Dec. 1972)

GUIDE70: "Data base systems requirements", Joint GUIDE-SHARE Data Base Requirements Group, (Nov. 1970)

HAMM72: "On the maximization of a pseudo-Boolean function", P. L. Hammer, U. N. Peled, JACM, Vol. 19, No. 2, (Apr. 1972)

HEAPS72: "Storage analysis of a compression coding for document data bases", INFOR Journal, Vol. 10, No. 1, (Feb. 1972)

HEATH71: "Unacceptable file operations in a relational data base", I.J.Heath, SIGFIDET 1971

HSIAO70: "A formal system for information retrieval from files", D. Hsiao, F. Harary, CACM, Vol. 13, No. 2, (Feb. 1970)

JARD68: "The construction of hierarchical and non-hierarchical classifications", N. Jardine, R. Sibson, Computer Journal, Vol. 10, (1968)

JARD71: "Choice of methods for automatic classification", N. Jardine, R. Sibson, Computer Journal, Vol. 14, No. 4, (Nov. 1971)

KOH70: "Switching and finite automata theory", Z.Kohavi, McGraw-Hill Book Co. (1970)

KNOTT71: "Expandable open addressing hash table storage and retrieval", G.D.Knott, SIGFIDET 1971

KNUTH70: "Optimum binary search trees", Donald E. Knuth, Technical Report CS 149, Computer Science Dept., Stanford University, (distributed by NTIS: PB-188-748),(Jan. 1970)

LANCE66: "Computer programs for hierarchical polythetic classification ("similarily analysis")", G. N. Lance, W. T. Williams, Computer Journal Vol. 9 (1966)

LANCE67A: "A general theory of classificatory sorting strategies, 1. Hierarchical systems" G. N. Lance, W. T. Williams, Computer Journal, Vol. 9 (1967)

LANCE67B: "A general theory of classificatory sorting stragegies, 2. Clustering systems", G. N. Lance, W. T. Williams, Computer Journal, Vol. 10, (1967)

LANCE68: "Note on a new information-statistic classificatory program", G. N. Lance, W. T. Williams, Computer Journal, Vol. (1968)

LAWL69: "Branch-and-bound methods: a survey", E. L. Lawler, D. E. Wood, Operations Research, Vol. 14, No. 4, (1969)

LEFK69: "File structures for on-line systems", D. Lefkovitz, Spartan Books (1969)

LING72: "On the theory and construction of k-clusters" R. F. Ling, Computer Journal, Vol. 15, No. 4, (Nov. 1972)

LOWE68: "The influence of data base characteristics and usage on direct-access file organization", T. C. Lowe, JACM, Vol. 15, No. 4, (Oct. 1968)

LUM70: "Multi-attribute retrieval with combined indexes", V. Y. Lum, CACM, Vol. 13, No. 11, (Nov. 1970)

LUM71: "Key-to-address transform techniques: a fundamental performance study on large formatted files", V. Y. Lum, P.S.T. Yuen, M. Dodd, CACM, Vol. 14, No. 4, (Apr. 1971)

LUM72: "Additional results on key-to-address transform techniques: a fundamental performance study on large existing formatted files", V. Y. Lum, P.S.T. Yuen, CACM, Vol. 15, No. 11, (Nov. 1972)

LUM73: "General performance analysis of key-to-address transformation methods using an abstract file concept", V. Y. Lum, CACM, Vol. 26, No. 10, (Oct. 1973)

MAN73: "A model for keyword based file structures and access", F. Manola, D. Hsiao, NRL Memorandum report 2544 (distributed by NTIS: AD-745-409), (Jan. 1973)

MAR67: "Automatic data compression", B. A. Marron, P. A. D. de Maine, CACM, Vol 10, No. 11, (Nov. 1967)

MAUR68: "An improved hash code for scatter storage", W. D. Maurer, CACM, Vol. 11, No. 11, (Nov. 1968)

MINSK71: "Multi-routine access to information stored on rotating-mass-storage devices", N. Minskey, Technical report 71-1, Institute of Technology, University of Minnesota, (Dec. 1971)

MINSK72: "Rotating storage devices as partially associative memories", N. Minsky, Technical report 72-4, Institute of Technology, University of Minnesota, (April 1972)

MOR68: "Scatter storage algorithms",Robert Morris, CACM, Vol.11, No.1, (Jan. 1968)

MORG72A: "Mathematical models of file processing: survey and classification", H. L. Morgan, S. R. Kennedy, Information Sciences Technical Report No. 3, California Institute of Technology (June 1972), (distributed by NTIS: AD-746-154)

MORG72B: "Optimal space allocation on disk storage devices", H. L. Morgan, Information Sciences Technical Report No. 4, California Institute of Technology (June 1972), (distributed by NTIS: AD-746-495)

MULL71: "Retrieval-update speed tradeoffs using combined indices", J. K. Mullin, CACM, Vol. 14, No. 12, (Dec. 1971)

NIEV72: "Binary search trees and file organization", J.Nievergelt, SIGFIDET 1972

NIEV73: "Upper bounds for the total path length of binary trees", J.Nievergelt, C.K.Wong, JACM, Vol.20, No.1, (Jan. 1973)

PALM73: "Whose data base for us?", Ian Palmer, Computing, (Nov. 1, 1973), pp. 10-13

PATT69: "Variable length tree structures having minimum average search time", Y. N. Patt, CACM, Vol. 12, No. 2, (Feb. 1969)

PRYW63: "The organization of a multi-list type associative memory", N.S.Prywes, H.J.Gray, IEEE Transactions on Communications and Electronics, No. 68 (Sept. 1963)

RAM70: "Optimization of memory hierarchies in multi-programmed systems", C. V. Ramamoorhty, K. M. Chandy, JACM, Vol. 17, No. 3, (July 1970)

RICK72: "Design considerations for a Boolean search system with automatic relevance feedback processing", J. T. Rickman, Proc. ACM 1972

SALT63: "Associative document retrieval techniques using bibliographic information", G.Salton, JACM, (Oct. 1963)

SALT69: "Information storage and retrieval", G.Salton, Scientific Report No. ISR-16, Dept. of Computer Science, Cornell University, (Sept. 1969)

SENK73: "Data structures and accessing in data base systems", M. E. Senko, E. B. Altman, M. M.

Astrahan, P. L. Fehder, IBM System Journal, Vol. 12, No. 1, (Jan. 1973)

SEV72: "Performance evaluation of file organizations through modelling", D. G. Severence, A. G. Meyton, Proc. ACM 1972.

SHAP69: "A handbook on file structuring", R.M.Shapiro, H.Saint, R.F.Millstein, A.W.Holt, S.Warshall, L.Sampliner, Rome Air Development Centre (TR-69-313),(distributed by NTIS: AD-697-025)

SHEP68: "Cluster analysis on the Atlas computer", M. J. Shepherd, A. J. Willmott, Computer Journal, (1968)

SHN73: "Optimum data base reorganization points", B. Shneiderman, CACM, Vol. 16, No. 6, (June 1973)

SIB73: "SLINK: An optimally efficient algorithm for the single-link cluster method", R. Sibson, Computer Journal, Vol. 16, No. 1, (Feb. 1973)

STAMP71: "Some ways of measuring information", P. K. Stamper, Computer Bulletin, Vol. 15, No. 17, (Dec. 1971)

STONE72: "Retrieval efficiency using combined indices", M.Stonebraker, SIGFIDET 1972

SUSS63: "Use of tree structures for processing files", E.H.Sussenguth Jr., CACM, Vol.6, No.3, (May 1963)

VANR71: "An algorithm for information structuring and retrieval", C. J. van Rijsbergen, Computer Journal, Vol. 14, No. 4, (Nov. 1971)

WALK72: "Hybrid trees: a data structure for lists of keys", W.A.Walker, C.C.Gotlieb, SIGFIDET 1972

WALL68: "An information measure for classification", C. S. Wallace, D. M. Boulton, Computer Journal, (1968)

WEXEL67: "The MULTILANG on-line programming systems", R. L. Wexelblat, H. A. Freedman, AFIPS Conference Proc., Vol. 30 (1967), Spring Joint Computer Conference

WONG71: "Canonical structure in attribute based file organization", E. Wong, T. C. Chiang, CACM, Vol. 14, No. 9, (Sept. 1971)

WOOD68: "Switching theory",P.E.Wood, McGraw-Hill Book Co., (1968)

ZIS73: "Boolean minimization",D.Zissos,I.G.Duncan,
Computer Journal, Vol.16, No.2 (May 1973)