QUERY PROCESSING IN A MULTIMEDIA DOCUMENT SYSTEM

E. Bertino, S. Gibbs, F. Rabitti

Istituto di Elaborazione della Informazione
Consiglio Nazionale delle Ricerche
Via S.Maria 46, Pisa (Italy)

1.0 INTRODUCTION

Multimedia Information Technology has became a rapidly growing field of investigation within the last few years and several research activities concerning various aspects of multimedia information management have been reported [COM85], [IEEE84]. The challenge arises from the capability of providing an integrated, homogeneous way to access, process and exchange multimedia information objects. A multimedia information object, which we will also refer to as a multimedia document, may contain attributes, i.e. formatted data, text, images, graphics, as well as voice annotations. The name multimedia means that such objects can be perceived, created, and retrieved using different types of media.

The integrated management of multimedia documents offers great potential to increase the productivity of people in application environments such as offices, military applications, CAD/CAM, and libraries [CHRI85]. The development of a multimedia system involves several issues. Such a system should in fact provide capabilities to:

- Display, create and manipulate multimedia documents; issues here concern document presentation and rendition as well as the development of multimedia editors.

- Transmit and receive multimedia documents over both local and long-haul geographic computer networks; an important issue here concerns the definition of standards for multimedia information exchange.

- Store and retrieve multimedia documents; some of the issues here concern storage requirements, since multimedia documents are large in size, and powerful retrieval capabilities.

Another important issue concerns the definition of suitable conceptual models since traditional models used in data base systems do not provide enough flexibility in modelling multimedia objects [RABI85]. A multimedia document is a collection of components which contains the different types of multimedia information and may be further structured in terms of other components (such as the body of a paper that is composed of sections and paragraphs and contains images and attributes embedded in text). These complex structures, which can greatly vary from one document instance to another, cannot be adequatly described with the structuring mechanisms of the traditional data models.

Previous work on multimedia documents has been primarily concerned with document manipulation, presentation, and transmission. This is the case for instance of Diamond [THOM85], a system that allows the users to create, edit, manage, and transmit documents containing text, graphics, image, voice, and electronic spreadsheets. Some work has also been reported dealing with the problem of representing the structure of multimedia documents. In [GIBB83] a data model for office objects is presented. It is based on the object-oriented modelling approach, and is focused on more

dynamic aspects, such as data object presentation, manipulation and distribution.

A sophisticated model for multimedia documents is described in [BARB85]. In this model different document structures coexist in the same documents in order to support the different functions of editing (logical structure), presentation (layout structure) and retrieval (conceptual structure). Also some standards for document exchange, formatting and editing are under definition by international organizations [ISO84], [HORA85], [POST82]. The area of filing and retrieving multimedia documents, though, has not been explored as throughly.

A survey presented in [MURP83] has shown that almost all systems surveyed have facilities for storing and retrieving electronic documents. However the systems described present serious drawbacks for storing and retrieving of multimedia documents. If the documents are filed directly at the user's workstations one is likely to exceed the storage space, because of the large space requirements of multimedia documents. Moreover the distributed access of these documents can be a problem. Locating the document filing functions on a host computer has the advantages of exploiting a powerful operating environment. The available operating systems can support a variety of software for database and information retrieval systems. However these "traditional" tools may not be adequate in handling multimedia documents containing different types of data, and the performance obtained for large volumes of documents compared to the cost would probably be unacceptable. The use of a file server over a network allows the workstations to share the filing services [SVOB84]. However, if it is a general purpose server oriented to files handling, then multimedia documents can be seen only as ordinary files. In which case the only type of document retrieval would be by address or name.

The advantages of a network file server could be enhanced if the server is specialized for the filing and retrieval of multimedia documents. With this approach, it is possible to target the hardware and software choices towards the specific task of document storage and retrieval. In particular it would be possible to use high capacity storage devices such as optical disks.

A distributed system designed on this philosophy is presented in [BERT85] and [BERT86]. The system consists of a number of autonomous subsystems called multimedia document servers and a number of client subsystems. The system provides different types of filing capabilities: dynamic filing dealing with documents that are private to a client and that are under modification; current filing dealing with updatable shared documents; archive filing dealing with stable shared documents. In order to provide adequate storage capacity the servers have been designed so that they can support optical storage devices in addition to magnetic devices. Another important feature of the system is that it supports a the document model described in [BARB85]. Document retrieval is based on the conceptual structures as defined in this document model.

The internal structure of a document server consists of a certain number of components that are described in detail in

[BERT86]. Among these the storage subsystem provides access methods for document retrieval and allows the storage of large data values while the translator maps document level operations onto the data structures of the storage subsystem. Query processing in such a server is complicated by the fact that both magnetic and optical storage may coexist on a server. Since magnetic and optical storage have different organizations, different storage models must be built and used for query optimization.

In the present paper we discuss query processing in such an environment. The remainder of this paper is organized as follows. In section 2 we describe the query language and we briefly survey the conceptual document model. In section 3 we describe the server storage organization. In sections 4 and 5 we describe respectively the statistics and cost functions used in query optimization. In section 6 we discuss the various steps in query processing. Finally in section 7 we present query optimization strategies.

2.0 THE QUERY LANGUAGE

Retrieval of documents in our system is based on the document model described in [RABI85], [BAR885], and [RABI86].

The structure of multimedia documents is more complex than that of the objects usually managed in form processing systems [GEHA82] or information retrieval systems [SALT83]. In order to support different operations (i.e. editing, presentation, retrieval) the document model supports several structural descriptions of a multimedia document. The logical structure determines how logical components, such as sections and paragraphs, are related [HORA85]. The layout structure describes how document components are arranged on an output device at presentation time [HORA85]. There may be couplings between logical and layout structures (see, for instance, the template concept in [GIB883]).

For many operations, such as query specification by content and document creation, it is important to see a document in terms of its conceptual components. A conceptual component is a document component which has some commonly understood function in the organization. The associated document structure is called the conceptual structure. An example of a conceptual structure could be a "meeting announcement letter" containing a conceptual component called a "meeting subject". The conceptual structure is used for query specification since conceptual components are more meaningful to the user than logical or physical components and, also, the conceptual structure is often less complex than the logical and layout structures.

Since a document may go through successive processing operations after its creation, it is not sufficient to simply store the values forming the document, one must also store structural information. The document model adopts a standardized representation based on ODA (the Office Document Architecture, a standard under definition by ISO, ECMA and CCITT [ECMA85] [ISO84]) for the logical and layout structures.

Document retrieval by content is greatly enhanced if the conceptual role of document components can be described. For this reason, in the adopted document model the conceptual document structure has been introduced. Document type definition is essential for the retrieval. However, since multimedia document structures tend to differ greatly from instance to instance, the kind of database schemata that arise with strongly typed data models cannot be used for document type definition. For this reason, a weakly typed data model, aimed at the flexibility required in multimedia document definition, has been adopted. In this model, a weak type is the specification of the common conceptual components within a set of documents. (An instance of a weak type may have a much more complex and detailed structure than that specified in the definition of the type.) This concept of weak type allows the definition of types at different levels of detail: an "is-a" hierarchy can be constructed among weak types.

A type definition mechanism is useful for the storage and retrieval of data objects when the ratio between instances and types is high. In this case it is possible to use the structure specified within the type to build efficient access methods and as an aid in query formulation. There is little regularity in the logical and layout structure of multimedia documents. However the conceptual structure is more uniform and so is suitable for forming document types.

2.1 Basic Concepts of the Document Model

A document's conceptual structure is specified by mean of a Structure Tree. A leaf node in the structure tree is a node with no exiting edges. In the present paper we will refer to the nodes of the structure tree as **conceptual components**, components corresponding to non-leaf nodes in the structure tree will be referenced to as **intermediate conceptual components**. The symbol * attached to a node means that several instances of the corresponding conceptual component may appear in a document instance. We refer to such components as **multi-valued components**.

A simple example is given in fig.1. In the example RECEIVER is a multi-valued conceptual component. It should be noticed also that the conceptual components NAME and ADDRESS appear in two subtrees having as roots respectively the conceptual components RECEIVER and SENDER.

From the example it can be seen that there are basically two ways of referencing to components:

- by a "simple" name,
- by a path-name.

A path-name has in general the form

NAME<1> {.|*} NAME<2> {.|*} ....NAME<n-1> {.|*} NAME<n> [1]

where each NAME<i> is a simple name.

The path-name specifies that the conceptual component being referenced is the component having as "simple" name NAME<n> and which is contained within the conceptual component whose name is NAME<n-1>. The conceptual component NAME<n-1> is in turn contained in NAME<n-2> and so forth. Component names within a path-name can be separated by either a "." or a "*". When a "." is used this means that the conceptual component of the left side of the "." contains

---

[1] Throughout this paper, for XXX<i> we intend XXX with index i. This representation can be nested:
XXX<i<j>> means XXX with index i, i having index j.

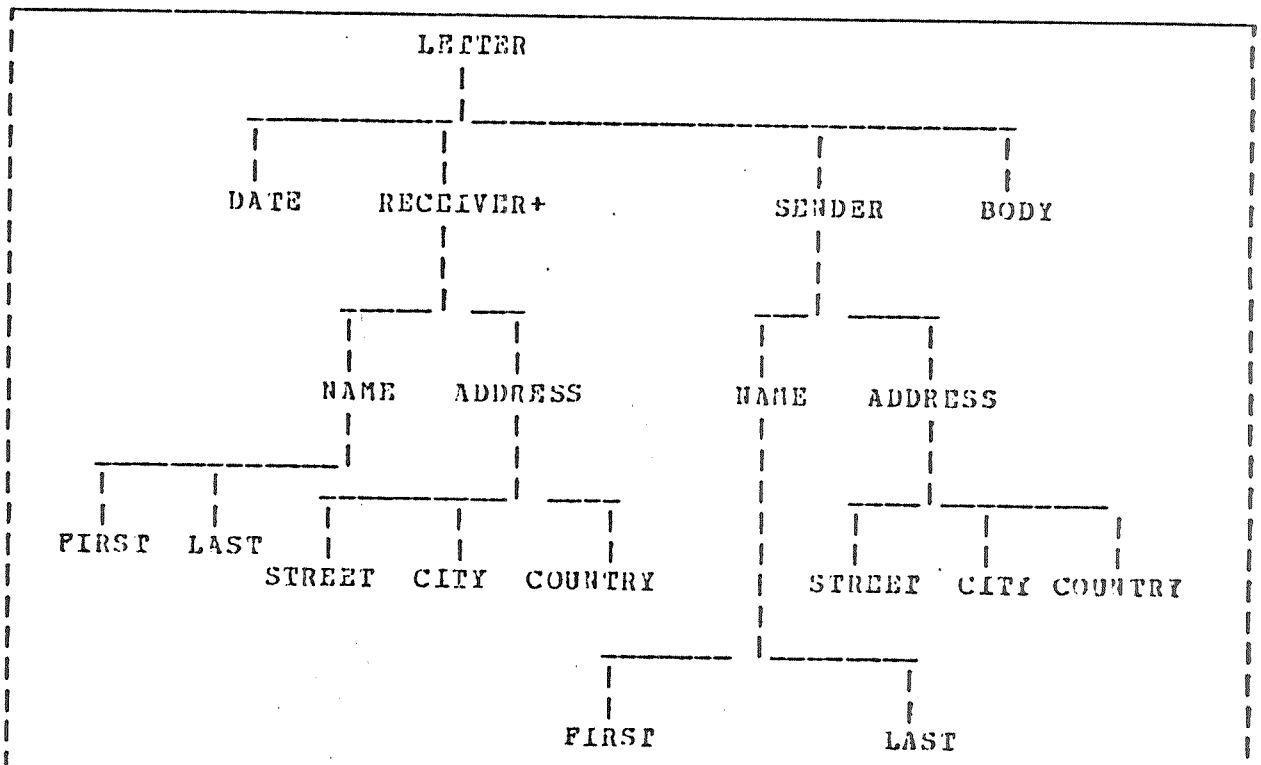


Fig.1 Conceptual Structure for the Document type LETTER

directly the component of the right. When a "*" is used there may be one or more intermediate components between the component on the left side and the one on the right. Examples of path-names are the following:

- SENDER.NAME
  In this case the component referenced is the one having as simple name NAME and which is contained in the conceptual component SENDER.

- RECEIVER*COUNTRY
  In this case the component referenced is the one having as simple name COUNTRY and which is contained in RECEIVER. COUNTRY however is not directly contained in RECEIVER since between the two there is the intermediate component ADDRESS.

2.2 Language Definition

In general queries may have conditions on both the content and the conceptual structure of documents. Expressing conditions on the document conceptual structure asks for documents having the conceptual component whose name is specified in the condition.

In addition, a document conceptual type can be specified in the query. In such case, the conditions expressed in the query apply to the documents belonging to that particular conceptual type. If the document type indicated in the query has subtypes, then the query applies to all the documents having as type one of these subtypes. When no type is specified, the query will apply to all possible document types.

Finally a scope can be indicated in the query. This allows restricting the query to a particular set of documents. Usually this set of documents is a document collection retrieved by a previous query.

In general, then, a query can be seen as composed of three clauses:

SCOPE The scope restricts the query to a given set of documents. The scope can be specified either by giving a collection identifier or by giving an explicit set of document identifiers. This clause is optional.

TYPE The TYPE specifies a document conceptual type at any level at the type hierarchy. This clause is optional. When it is missing the query applies to all document types.

COND The COND clause is a boolean combination of a set of simple conditions, or predicates, that must be satisfied by the documents retrieved. We discuss in the following the types of simple conditions that can be specified in the language.

2.2.1 CONDITIONS

In our language conditions are usually expressed against conceptual components of documents, that is a condition has in general the form:

<component> <restriction>.

Where <component> is the name (or path_name) of a conceptual component and <restriction> is in general an operator followed by an expression. We discuss restrictions in the next subsection.

However, to implement content addressability for the text part of documents it is important to allow text restrictions that do not refer to a particular conceptual component. Such conditions will be matched against all text components of the documents searched. In this case a simple condition has the form:

<restriction>

where <restriction> is one of the allowed restrictions for text.

Another important point concerns multi-valued components. Documents may contain repeating values for a conceptual component. In

such a case, the query language must allow one to express the fact that a restriction applies to either all the values of the multivalued components or at least one value. To this purpose, we have introduced the two quantifiers ALL and SOME. A condition involving quantifiers has the form:

{ALL|SOME} <component> <restriction>.

Where <component> is the name (or path name) of the multivalued conceptual component. Also we have introduced for the multivalued components the operator IS IN. A condition containing the IS IN operator has the form:

<component_name> IS IN <set-expression>.

The semantics of this condition is that the set of values of the multivalued component must be a subset of the given set.

We refer to a condition containing quantifiers or the IS IN operator as a **multivalued atomic condition**. Any other type of condition is called a **singlevalued atomic condition**.

As we said previously a condition has the form

<component> <restriction>

It should be noticed that the component name (or path-name) may refer to a conceptual component which is contained in several conceptual components. For instance, in the example in fig.1, we could have condition of the form:

NAME <restriction>

The restriction applies to both the components whose path-names are SENDER.NAME and RECEIVER.NAME. The problem is to decide how such a condition is satisfied. There are two possibilities:

1. NAME <restriction> = True if
   (SENDER.NAME <restriction> = True) AND
   (Some(RECEIVER.NAME <restriction> = True))

2. NAME <restriction> = True if
   (SENDER.NAME <restriction> = True) OR (Some(RECEIVER.NAME <restriction> = TRUE))

We have chosen to use the second solution since it seems more flexible for queries which are typeless or refer to very general types.

In addition to the previous types of conditions, the language must allow conditions on the existence of conceptual components within documents. This allows expressing queries on the conceptual structure of documents. Therefore we have defined the operator WITH. A condition containing the WITH operator has the form:

WITH <component>.

This condition expresses the fact that the component whose name (or pathname) is given, must be a conceptual component of the documents to be retrieved. To express conditions in which it is required that a conceptual component having name NAME<i> is contained in a conceptual component having name NAME(j) the path-name NAME<i>*NAME<j> is used.

2.2.2 RESTRICTIONS

Four types of restrictions have been defined: numeric, string, text, date, and component restrictions.

Numeric restrictions have the form:

<rel_op> <numeric_expr>

where <rel_op> is one of the following operators:

=, !=, <, >, <=, >=, BETWEEN.

The BETWEEN operator is used to express range conditions.

String restrictions have the form:

[<rel_op>|LIKE] <string_expr>

where the <rel_op> is one of the operators we have listed before and the LIKE operator is used for partial match. In particular when the LIKE operator is used, the string expression may contain any characters, with special meaning reserved reserved for the characters "_" and "%". The "_" character represents "any single character". The "%" character represents "any string of zero or more characters". These two special characters may be used, in any combination, in the string expression.

Text restrictions have the form:

CONTAINS <string_expr_list> [<distance>]

where <string_expr_list> is a list of strings. This restriction specifies that the text must contain the given strings. The <distance> clause specifies the distance between the given strings.

Finally component restrictions have the form:

<rel_op> <comp>

where <rel_op> is one of the relational operators and <comp> is the name (or pathname) of a conceptual component. Restrictions of this type are used to specify for example that two conceptual component have the same value.

The query language grammar is defined in "A.0 Append. A - The Query Language Grammar" pag. 46.

3.0 PHYSICAL STORAGE ORGANIZATION

In describing secondary storage we will use the following terms:

devices - a device is either magnetic or optical. Magnetic devices consist of a number of cylinders, each containing **tracks** which in turn contain **sectors**. An optical disk simply contains a sequence of sectors. Optical disks may be arranged in a "juke-box" in which case one disk is mounted and the remaining are unmounted. To read or write an optical disk it must be mounted.

segments - devices may be divided into segments. A segment is a set of extents. An optical segment may be **open** or **frozen**. An optical segment is frozen once no more writing is allowed on the segment, this usually occurs when the free space drops below some minimum value. If writing is possible, the segment is open.

extents - an extent is a physically contiguous region of secondary storage. An optical extent is a set of sequential sectors, a magnetic extent is a cylinder. Each segment may be described in terms of the extents it contains.

blocks - each extent consists of a sequence of fixed size blocks.

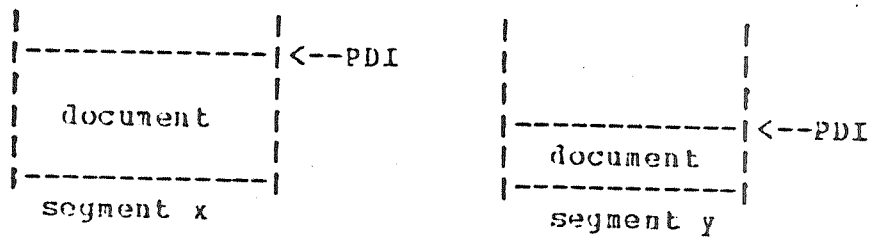
block pointer - a block pointer locates a block within a segment. It is the offset from the start of the segment.

files - a file is a group of blocks. All blocks in a file are on the same segment. The manner in which the blocks are grouped will depend on the file organization. There are three file organizations used: sequential, linked-list, and B-tree. In the first, blocks are grouped sequentially, in the last two, by explicit links. Links are implemented using block pointers. The pointer to the first block of a file is called a **file pointer**.

There are four possible uses of a segment: "bulk storage", text signatures, document indexes, and system tables.

3.1 Bulk Storage Segments

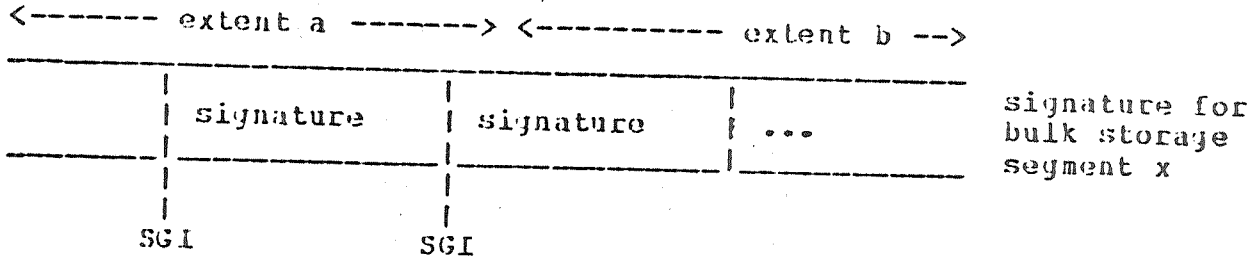
Bulk storage refers to those segments used to store document instances. Documents are stored in bulk storage sequentially. The location of a document is specified by a PDI, a **physical document identifier.** A PDI consists of a <segment number, block pointer> pair. (Documents are stored on block boundaries.) The following figure depicts this organization:



In addition to a unique PDI, each document also possesses a unique LDI or **logical document identifier.** LDIs are assigned by the system on document insertion.

3.2 Text Signature Segments

Associated with each bulk storage segment is one signature segment used to hold text signatures for accessing the documents on the bulk storage segment. A signature segment contains a single signature file. The structure of a signature file is as follows:



A signature file consists of a sequential list of signatures. The signature of a particular document is indicated by its SGI, or **signature identifier.** An SGI is a <segment number, block pointer> pair. (Signatures are stored on block boundaries.)

The signature of a document consists of a header followed by the signatures of the document text blocks. The header contains the LDI, total signature length, and a delete flag.

Each text signature file is stored in a signature segment which contains the signature of the text portions of the documents stored in the same bulk storage segment. A signature segment can be stored either on a magnetic or on an optical device.

Different signature techniques have been proposed in the literature. The word signature [TSIC83] and the superimposed coding [CHRI84a] are very flexible with respect to document insertion, deletion and update. However, they require a complete scan of the signature file in case of a generic text query. The bit slicing of superimposed codes [ROBE79] requires access of a limited number of signature blocks also in case of generic text queries. However, in case of document modification/insertion it implies an expensive reorganization of the signature file. Between word signature and superimposed coding, the latter performs better in terms of space overhead and CPU consumption [RABI84]. With superimposed coding the ordering of words in a signature block is lost. However, if required in the text query, the word order can be checked when the document is retrieved from the bulk storage segment.

Given these considerations, it appears that the superimposed coding technique (SCT) is more suitable for magnetic signature segments, while the bit-slicing of superimposed codes (BST) is more suitable for optical signature segments.

The storage occupation for SCT and BST is the same, the only difference is in the size of signature blocks and in the meaning of the bits contained.

For the moment we assume that signatures are stored sequentially in the signature file. The usage of bit-slicing technique is however under consideration.

3.3 Index Segments

Each index segment contains a number of document indexes. A document index consists of two files: a B-tree file and a inverted file. The B-tree file contains pointers to an inverted file while the inverted file contains document component values and the LDIs of the documents containing those component values. Records in the inverted file are ordered by key value.

B-tree files are not stored contiguously on either magnetic and optical storage. The blocks of an inverted file are stored contiguously on optical storage, while they are not stored contiguously on magnetic storage.

The magnetic index segments together form a composite index which indexes all documents on all segments. An optical index segment, however, indexes only those documents within a single (optical, frozen) segment.

3.4 System Table Segments

Finally certain magnetic segments are reserved for system tables and temporary file space.

We divide the control information used by the storage subsystem into two parts: internal data structures and system tables. Internal data structures are used only within the storage subsystem and are not visible to other server components. The internal data structures are used to store information concerning transaction control, device configuration, and segment and extent layout. We will not further specify these structures here. System tables contain shared and persistent information. System tables are maintained by the storage subsystem and can be accessed by the higher level components of the document server. Examples of such information are conceptual type definitions and statistics that are used by the query processor.

3.5 Clusters

The concept of a **cluster** is useful in describing the grouping of segments on optical storage. We will assume that a frozen optical bulk storage segment has a single signature segment and a single index segment. Such a group of three segments forms a cluster. Clusters are meant to be self-contained in that they contain a set of documents plus access information for those documents.

A cluster is formed when a bulk storage segment is frozen. The cluster's signature segment is formed by either copying the original signature segment (if a magnetic segment was being used) or calculating the signature directly from the document text (if a different signature encoding is used). The cluster's index segment is formed by extracting indexes from the composite index and writing these to an optical segment.

3.6 Access Paths

In our system we have three types of access paths:

- Document Access
  The access path consists of scanning the set of document instances. It may take as an argument a set of PDI. In this case we have random accesses to the set of documents. If it has no arguments the whole set of documents is scanned. In this case we have a serial access.

- Index Access
  An index can be scanned by using the B-tree or by using directly the inverted file. This last type of scan is used for instance to evaluate predicates such as <,<=. In this case the inverted file scanned from the beginning until a record is found that does not satisfy the search condition. An index scan has as input parameter a search condition consisting of:

  the index name;

  an index operator (OP); the operator can be one of =, <,>,<=,>=;

  a key value (KV);

  The result of an index scan is the set of LDIs of the documents satisfying the search condition.

- Signature Access
  A signature scan has the following input parameters:

  a set of text conditions {T<1>,T<2>,...T<n>}
  (the form of the text conditions is defined below)

  a set of document identifiers (optional).

  The result of a signature scan is for each T<i> the set of LDIs of documents that satisfy T<i>.

  A text condition has the following form:
  {m, S_E_L}
  where S_E_L is a set of string expression lists and m is the number of string expression lists contained in the set S_E_L. The documents that satisfy a text condition are the ones that satisfy at least one of the string expression lists in the set S_E_L. A string expression list is defined in the query language grammar (see Appendix A). A document satisfies a string expression list if it contains all the strings contained in this string expression list.

  It should be noticed that by passing to the signature handler several string expression lists one can evaluate all the text restrictions within a query with only one signature scan.

The costs of the various access paths are presented in Section 5.

3.7 Storage Subsystem Parameters

The essential parameters necessary to describe the storage devices inside the storage subsystem are:

- For magnetic media:

  MSK = Seek time for locating a physical block (in sec.)

  MTR = Disk transfer rate (in bytes/sec.)

- For optical media:

  OSS = Seek time for locating a physical block (in sec.)

  OTR = Disk transfer rate (in bytes/sec.)

  OMT = Disk mounting time (in sec.), in the case we have a juke-box which allows the switching of optical disks.

4.0 STATISTICS

During query processing statistical information is used to estimate the selectivity of the various restrictions appearing in a given query. Statistics are collected and stored for each document cluster. It should be noticed that separate statistics are needed for each cluster since document characteristics may vary from cluster to cluster. As a consequence, the optimal execution of a given query, may vary from cluster to cluster.

The following statistics are used to evaluate the selectivity of the various restrictions:

- SD = Average document size in bytes
- STD = Average size in bytes of the text part of a document
- For each conceptual document type, T, the number of documents having type T. We indicate this as NDI(T)
- For each conceptual document type, T, the average document size. We indicate this as SD(T).
- For each conceptual document type, T, the average size of the text part of the document. We indicate this as STD(T).
- For each bulk storage segment, S, the number of documents contained. We indicate this as NDB(S)
- For each multivalued conceptual component, c, the average number of occurrences of that component in a document. We indicate this as n(c).
- For each index I:

  NA(I) = number of distinct keys in the index

  HV(I) = highest key value

  LV(I) = lowest key value

  NL(I) = number of blocks in the inverted file

  NI(I) = index cardinality

  U(I) = 1 if the index is unique else = 0

4.1 Statistics for text

It is important to identify the target to which the various text statistics refer. We call this target a Text Set, which may be a set of text documents or a set of text document components. In relation with our storage subsystem, a text set is defined as the text which is represented in a single signature segment.

With respect to a text set X, the statistical parameters relevant for determining the text access times are:

- W[X] number of words in X
- DW[X] number of distinct words in X
- NCW[X] number of non common words in X
- CW[X] number of common words in X
- NCDW[X] number of non common distinct words in X
- CDW[X] number of common distinct words in X
- CWL number of common distinct words specified in the stop-word list for the application.

Not all these parameters are independent. The following relationships hold:

- W[X] = CW[X] + NCW[X]
- DW[X] = CDW[X] + NCDW[X]

It is usually too expensive to keep all these statistics for each text set X. In general, it is more convenient to define for X the following functions:

- DW[X] = f(W[X])
- CW[X] = g(W[X])
- NCDW[X] = h(NCW[X])

If we can determine f(), g() and h() for X, we are able to determine all the statistical parameters for X knowing only W[X].

For the function g(), we can assume [RABI84]
CW[X] = CWPER * W[X]
where the constant CWPER represents the percentage of common words and is determined by the choice of the stop-word list for the application.

The most crucial part, is estimation of the function f(). In order to predict the performance of an access method it is essential to know the number of unique words for the text set and any portion of it. For example, in computing the performance of signa-

ture techniques it is important to estimate the number of distinct words in a page or block. In [RABI84] it is shown that no analytic function can closely approximate DW[X] for very different text sizes W[X]. Two main factors, reflecting the text set characteristics, determine the function f():

1. DENS = document density
   represents the average variety of words in each text component. This quantity is sensitive to the type of the document and the writing style. If DENS->1, each word is unique, while if DENS->0 there is only one repeated word.

2. FOUT = document fanout
   represents the average variety of words among the different text components. FOUT reflects correlation in topic among documents. If FOUT->1, each document contains different words, while if FOUT->0, documents contain the same words.

In [RABI84], a way to approximate the parameters DENS and FOUT has been proposed. It is easy to compute, since it consists in counting words (W) and distinct words (DW) for two different sample subsets of the text set X.

1. For the parameter DENS, we refer to a sample subset X1 of X, with "standard" characteristics in X. For example, its length is adjusted so that W[X1] corresponds to the average length of document or text components in X. We define L1=W[X1]. We impose:
   DENS = DW[X1] / W[X1]

2. For the parameter FOUT, we refer to a simple subset X2 of X, such that its length W[X2] corresponds to the average length of 1000 documents or text components in X (i.e. 1000*W[X1]). Notice that FOUT can be computed only when the size of X is large enougth. We define L2=W[X2]. We impose:
   FOUT = ln(DW[X2]) / ln(W[X2]) [2]

In [RABI84] it is observed that the function f() presents different behavior on three intervals for W[SX], where SX is a generic subset of X. The intervals are: below L1, from L1 to L2, and beyond L2. The influence of document density and document fanout on the function f(), differs in these intervals.

These functions are:

1. For W[SX] < L1
   f1(): DW[SX] = W[SX] / (1 + A1*W[SX])
   where A1 = (1/DENS) - 1.

2. For L1 < W[SX] < L2
   f2(): DW[SX] = B2 * W[SX]**A2
   where B2 = 10*DENS
   and A2 = FOUT - ln(B2)/ln(W[SX])

---

[2] In the following, for ln we mean the natural logarithm.

3. For W[SX] > L2
   f3(): DW[SX] = B3 * W[SX]**A3,
   where B3 = 100*DENS
   and A3 = FOUT - ln(B3)/ln(W[SX])

For the function h(), we notice that for a large text set X we have DW[X]>>CWL, so it is likely that all words in the stop-word list actually appear in the sample (i.e. CDW[X]=CWL). We have:
NCDW[X] = DW[X] - CWL

Otherwise, for a small size text set A it is probable that CDW[X]<CWL. In this case, the following approximation is acceptable [RABI84]:
CDW[X] = CWPER*0.5*DW[X]
thus, we have:
NCDW[X] = DW[X] * (1 - CWPER * 0.5)

4.2 Selectivity of predicates on attributes

Attribute selectivity is evaluated on the hypothesis that the values of each attribute are uniformly distributed and non-correlated. Even if these assumptions may provide a pessimistic cost estimations, as discussed in [CHRI84], they are generally assumed by query optimization algorithms of existing systems.

In general a predicate has the form:

<comp> <restriction>

where <comp> is the name of a conceptual component and <restriction> is an operator followed by an expression.

The selectivity of a predicate p is denoted by st(p) and it is estimated as is usually done in database systems. The selectivity of the various types of restrictions is listed in Appendix B.

It should be pointed out that since not all the conceptual document types have the same components, the estimate of the number of documents satisfying the a predicate p involving a conceptual component c is calculated as follows:

NDT(T) * st(p) if c is a singlevalued component

NDT(T) * (1 - ((1-st(p))**n(c)) [3]
if c is a multivalued component

where T is the highest document type in the type hierarchy having as conceptual component the one whose name appears in p. Therefore st(p) defines the selectivity of the predicate on the set of docu-

[3] x**y is read as x raised to y

ments belonging to type T. The selectivity of p on the entire set of documents is denoted by s(p) and it is calculated as follows:

s(p) = (NDT(T)/ND) * st(p) if c is singlevalued

s(p) = (NDT(T)/ND) * (1 - ((1-st(p))**n(c)) if c is a multivalued component.

4.3 Selectivity of predicates on text

In computing the selectivity of predicates on text, we assume the independence of the words in the text sets. In our case a text set can be identified as the text part of a set of documents whose signature is stored in the same signature segment. Otherwise, we could consider only one text set, comprehensive of all documents stored in the system. With these simplifications the text statistics we need to compute are W[X], DENS, FOUT.

We also assume the equiprobability of words, that is, the probability to find a word in a text set is equal to the average frequency of any word in the text set. More precise estimates would be possible if we keep track of the relative freqeuncy of each word in the text set [RABI84], but this would require maintaining a dictionary of distinct words. This is too expensive in our application environment.

Since in signature creation only non-common words are considered, we compute the selectivity based only on non-common words in the text set. We call pw the probability to find a specific word in a certain position of the text set X:

pw = NCDW[X]/NCW[X]
= (DW[X] - CWL) / W[X]*(1-CWPER)
where DW[X] = E(W[X]).

The probability to find a word in the subset SX of X is:
pw[SX] = 1 - (1-pw)**NCW[SX]
where NCW[X] = (1-CWPER)*W[X]

If we consider only one text set in the system, we can estimate the number of documents selected with one word predicate:
ND * (1 - (1-pw)**((1-CWPER) * (STD/(WLEN+1.2))))
where WLEN is the average word length in bytes and 1.2 is the overhead due to blanks and punctuations.
and, if the documents are restricted to the type T:
NDT(T) * (1 - (1-pw)**((1-CWPER) * (STD(T)/(WLEN+1.2))))

The probability of boolean combinations of text predicates p can be combined in the following manner, assuming the predicates are independent:

- P(p1 OR p2) = P(p1) + P(p2) - P(p1)*P(p2)

- P(p1 AND p2) = P(p1)*P(p2)
- P(NOT (p)) = 1-P(p)

5.0 COST FUNCTIONS

In the following we present the cost formulae for the various access paths. The cost functions are in terms of I/O. For each I/O operation the cost formulae consider both the seek and transfer time.

5.1 Cost Formulae for Document Access

Documents are stored in bulk storage segments and are identified by a PDI. The cost of accessing n documents depends on whether the storage segments where the documents are stored are on magnetic or optical devices.

- In case of magnetic disks, we have:
  COST = n*MSK + (n*SD)/MTR
- In case of optical disks, supposing that the n documents are spread into segments on m different optical disks, we have
  COST = m*OMT + n*OSK + (n*SD)/OTR

If the documents belong to a specific type T, SD(T) will be substituted for SD in the previous formulae.

5.2 Cost Formulae for Index Access

The cost formulae reported here are derived from [SELI79], [YAO79], [ASTR80]. In the following the notation log<z> is read as log with base z.

As described in Section 3 an index can be accessed by using the B-tree or by using directly the inverted file. Given an index I the costs for both cases are computed as follows:

Case 1

The index is unique and the index predicate is =. In this case at most one LDI is retrieved.

The cost for magnetic storage is given by:

C(I) = (log<z>NI(I) + 1) * (MSK+ BLK/MTR)

The cost for optical storage is given by:

$C(I) = (\log_z NI(I) + 1) * (OSK + BLK/OTR)$ where:

NI(I) is the index cardinality;

z is a number dependent upon the storage organization; it is a number such that a block holds between z and 2z keys (index elements) [BAYE72].

Case 2

The index is not unique and the index predicate is one of =, >, =>.

The cost for magnetic storage is given by:

$C(I) = (\log_z NI(I) + f * NL(I)) * (MSK + BLK/MTR)$

The cost for optical storage is given by:

$C(I) = \log_z NI(I) * (OSK + BLK/OTR) + OSK + f*NL(I)*(BLK/OTR)$

where f is the selectivity of the index predicate.

Case 3

The predicate is one of <,<=.

The cost for magnetic storage is given by:

$C(I) = f*NL(I) * (MSK + BLK/MTR)$

The cost for optical storage is given by:

$C(I) = OSK + f*NL(I)*(BLK/OTR)$

In the previous formulae BLK is the block size in bytes.

The main difference between the magnetic and optical storage is that in the first case the blocks of the inverted files are not contiguous therefore a seek operation is needed for each block access. In the second case inverted files are stored contiguously and only a single seek operation is needed.

5.3 Cost Formulas for Signature Access

The SCI signature is constructed in this manner: The text portion of each document is divided into one or more text subsets TS, each containing the same number of words (except the last TS). A signature block SIGN[TS] is composed of F bits. A word W is represented in a signature block by M bits. Common words are disregarded. A hashing technique is used to determine the M bits corresponding to each W. SIGN[TS] is created by initially setting all its F bits to "0", and then setting to "1" the bits corresponding to the words W

present in B. In this manner, only NCDW[TS] words are coded in SIGN[TS]: duplicate words in TS are represented once in SIGN[TS]. The SCT signature segment consists of the signatures of the TS's of documents contained in the corresponding bulk storage segment.

In order to query the SCT signature for a single word W (i.e. simple predicate), the same transformation is applied to W and the M corresponding bits are determined. Then all the indicated bits are checked in all the signature blocks SIGN[TS] by scanning the signature segment. If all M bits are set, W is assumed to appear in that TS of the document. In order to perform a query with a complex predicate (i.e. a boolean combination of several words) it is still necessary to scan the entire signature segment, only the processing in the core memory buffers becomes heavier.

In order to estimate the cost of text query processing for signatures the parameters M, F, and NTS (i.e. the number of TS's in the signature segment) must be derived. First, it necessary to fix the desired false drop probability (FDP), which gives the expected error rate for the signature technique (eg. 1/1000 or 1/100000). If W[TS] is the number of words at most contained in a TS, we have:

DW[TS] = f1(W[TS]) = W[TS]/(1+A1*W[TS])

where A1 is either relative to a particular signature segment or to the text set of all the documents in the system.

NCDW[TS] = (1-CWPER*0.5) * DW[TS]

M = | ln(FDP)/ln(2) |

F = | (M*NCDW[TS]) / ln(2) |

NTS = NDB(S) * (0.5 + (DTS / ((WLEN+1.2) * W[TS])))

where the constant 0.5 represents the average waste in the last TS of a document.

The storage overhead of the signature is:

(NTS * |F/8|) / (NDB(S) * DTS)

In case the bulk storage segment contains only documents of type T, DTS should be substituted by DTS(T).

We can now estimate the cost of computing simple and complex text predicates (on documents stored in a single bulk storage segment), using different access strategies. Let

- NE be the number of extents of the signature segment;
- NL be the number of LDI's of documents to which the query is to be restricted. Notice that if there is no such restriction, we impose NL=NDB(S).

Two strategies are possible:

1. Sequential signature scan

   COST = NE*MSK + (NTS * |F/8|) / MTR

   The parameters NL and NP do not appear in this cost formula. In fact, since only I/O cost is considered, SCT cost is independent from the number of words in the text predicate. Moreover, since all signature blocks are accessed, the cost is independent of NL.

2. Random signature access

```
COST = NSK*NSK + NL * (NTS * |F/B|) / (NDB(S)*NTR)
       + CADDR(NL)
```

where NSK = min (NL,NE)
CADDR(NL) is the cost of determining the signature addresses for a number NL of documents.

The number of disk seeks is estimated as the minimum between the number of documents and the number of extents.

LDI's are identifiers generated by the system, so they can be sequentially generated as natural numbers. Address tables for LDI's, in particular to determine for a given LDI the signature address SGI and the physical document address PDI, can be structured as sequential files in which the LDI is the absolute record number of the corresponding address.

With this assumption, we can compute the function CADDR(X) as:

```
CADDR(k) = b * (NSK + p*4 / NTR)
```

where each address takes 4 bytes, p is the blocking factor for these addresses (i.e. number of addresses per disk block). If m is the total number of blocks for the address table on disk, we can use the formula in [CARD75], as used in System R [ASTR80], to compute b:

```
b = m * (1 - (1 - 1/m)**k)
```

However, since this formula is based on the assumption that records (i.e. addresses) are selected with replacement, while in our case they cannot be selected more than once (supposing distinct LDI's), an error up to 36.8% can be introduced [WHAN83]. If a more precise estimation is required, the formula in [WHAN83] can be used, with a maximum error of 3.7%.

```
When k <= n - p:
  b = m * ( (1 - (1 - 1/m)**k)
       + (1/(p*m**2) * k*(k-1)/2 * (1 - 1/m)**(k-1))
       + (1.5/((m**3)*(p**4)) * k*(k-1)*(2k-1)/6
       * (1 -1/m)**(k-1)) )

When k > n - p:
  b = m
```

6.0 STEPS IN QUERY PROCESSING

The task of query processing can be divided in four main phases. In the first phase, some initial activities are performed such as parsing and catalog access. Also the query is modified in light of the type hierarchy. We call this the **preprocessing** phase. In the second phase, the set of document clusters that must be accessed is determined. Since document distribution on the various clusters is transparent to the applications, to solve a query it is necessary to determine which clusters contain documents that can potentially satisfy the query. We refer to this phase as **multi-cluster query resolution**. Once the set of clusters involved in the query is determined, for each cluster a query processing strategy is defined. We refer to this phase as **single-cluster query optimization**. Finally the **query execution** takes place by following the strategies defined at the previous step.

Before describing the various phases in detail it is useful to give a classification of the predicates that can occur in a query. Predicates can be divided into four classes depending how they can be checked:

1. Predicates on the structure
   These predicates are evaluated by accessing the system catalogs.

2. Index predicates
   These predicates are evaluated by using the indexes.

3. Text predicates
   These predicates are evaluated by means of signature scanning.

4. Residual predicates
   These are predicates that can be only evaluated by accessing the documents, that is predicates on components for which there are no access structures. This is the case for instance of attributes for which there are no indexes. Also predicates defined on live (spring) nodes belong to this class.

In the remainder of the discussion we use the following terms:

- **index query:** a query issued against the index segments by using the access paths provided by the index handler;
- **text query:** a query issued against the signature segments by using the access paths provided by the signature handler;
- **document query:** a query issued against the bulk storage segments by using the access paths provided by the bulk storage handler.

6.1 Preprocessing Phase

This phase consists of the following steps.

6.1.1 PARSING

The query is parsed by a conventional parser. Parsing verifies that the query has a correct syntax. The parser output is a query parse tree, which is augmented and modified by the subsequent steps in the query processing. In the parse tree the COND clause, that is the boolean combination of predicates, is expressed in Conjunctive Normal Form (CNF).

6.1.2 CATALOG ACCESS

During the catalog access information concerning the definitions for conceptual types and components are fetched. This information is stored in several tables (or other data structures) in main memory to be used in the subsequent steps in the query processing.

6.1.3 COMPONENT CHECKING

If a type is specified in the query then it is checked that the conceptual components present in the query belong to that type. Also each conceptual component name is expanded to its complete path name. If there are several paths corresponding to the given name (see for an example section 3) the condition C in which the component appears is substituted by a disjunction of conditions C1,...Cn, where n is the number of the path names. Each Ci has the same form as C, except that the name of the conceptual component appearing in C is substituted by the i-th path name.

If no type is specified the catalogs are accessed to determine the document types containing the conceptual components whose names appear in the query conditions. The list of these types is added to the query tree. If no document type exists containing such conceptual components the query results in an empty set and query processing stops.

6.2 Multi-cluster query resolution

The goal of this phase is to determine the clusters involved in a query. The set of clusters to be accessed may be restricted in different ways. For instance if a collection identifier has been specified the query can be restricted only to clusters storing documents belonging to the collection. Also conceptual document types specified in the query can be used to restrict the number of clusters to access for query resolution. In the following we describe the steps in multi-cluster query resolution.

6.2.1 SCOPE CLAUSE RESOLUTION

If a SCOPE clause is specified in the query by means of a collection identifier, then the LDIs for documents contained within this collection are retrieved. The set of clusters where such documents are stored can then be determined.

The result of scope resolution step is a set of pairs of the form <cluster-id, {LDI}> where cluster-id is a cluster identifier.

6.2.2 TYPE CHECKING

For each conceptual document type in the query the identifiers of the clusters storing documents of that type are retrieved. This information is obtained from system tables. This set of cluster identifiers is intersected with the set obtained by the previous step. If the intersection is empty query processing stops. If however the previous step has not been executed because no SCOPE clause has been specified in the query, then the set of clusters potentially useful for the query are all the clusters storing documents having the given types.

It should be pointed out that type checking is useful if documents are grouped on clusters on the base of their conceptual types. If however clusters contain documents of almost every type, then type checking is not so useful.

6.2.3 COMPOSITE INDEX CHECKING

In this step the set of clusters is restricted by evaluating some of the index predicates in the query against the composite indexes. It should be noticed that not all the index predicates in the query are eligible for use in multi-cluster query resolution. This happens when a index predicate is in OR with a text predicate or a residual predicate. In this case the index predicate cannot be used to reduce the set of clusters that must be accessed, since there

may be clusters containing documents that satisfy the query even if they don't satisfy this particular index predicate. Fig.2 shows the COND clause of a query together with the predicates eligible for multi-cluster query resolution.

After the predicates for multi-cluster query resolution have been selected, a set of index queries are generated and executed to evaluate these predicates.

The result of this step is a set of logical document identifiers (LDI's). For each LDI in this set, the corresponding physical address is retrieved from system catalogs. The physical address contains the cluster-id of the cluster where the document is located. The result of this phase is therefore a set of pairs of the form <cluster-id, {LDI}> where {LDI} is the set of identifiers for documents located in this cluster that have satisfied the index query. Also the original query is reduced by eliminating the index predicates that have have been evaluated in this phase. The query obtained is referred to as **reduced query.**

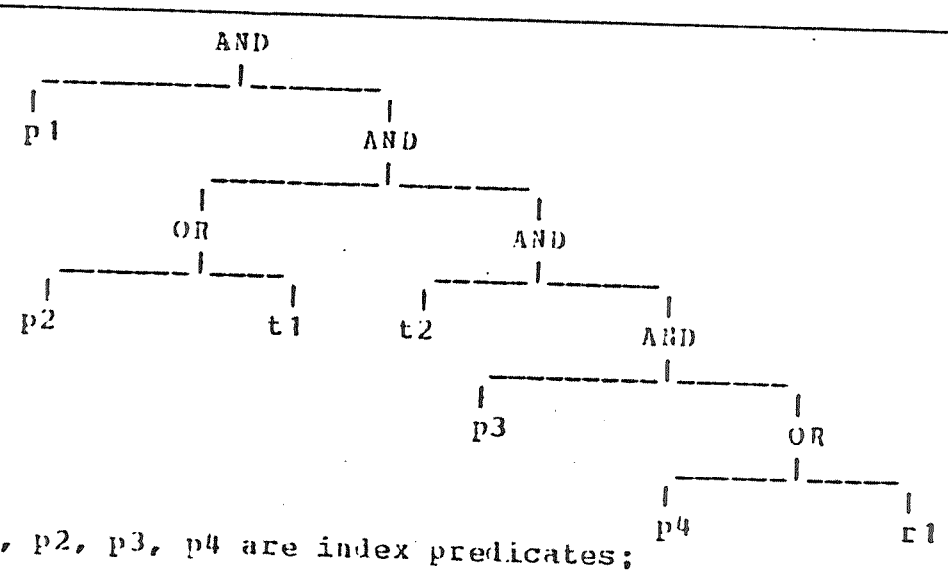
6.3 Single-cluster query optimization

For each cluster determined in the previous phase, a query execution strategy is defined. It should be pointed out that the given query must be optimized separately for each cluster involved, since the statistics used for query optimization may be different for different clusters.

The result of single-cluster query optimization is a set of index, text, and document queries. In addition a schedule is defined stating the order of execution for the various queries. It should be noted that index queries are generated only for the index predicates that have not been already evaluated during the multi-cluster query resolution. The single-cluster query optimization is described in detail in section 7.

6.4 Query Execution

For each cluster involved in the query, the corresponding segments are mounted, if they are not already mounted, and the query is executed following the strategy defined by the optimizer. Mounting a segment means mounting the disk where the segment is stored. The result of the query is a set of LDI's. The result of the original query is the union of the LDI's returned by all the single-cluster queries.



p1, p2, p3, p4 are index predicates;

t1, t2 are text restrictions;

r1 is a residual predicates.

The index predicates eligible for multi-cluster query resolutions are p1 and p3.

Reduced Query:

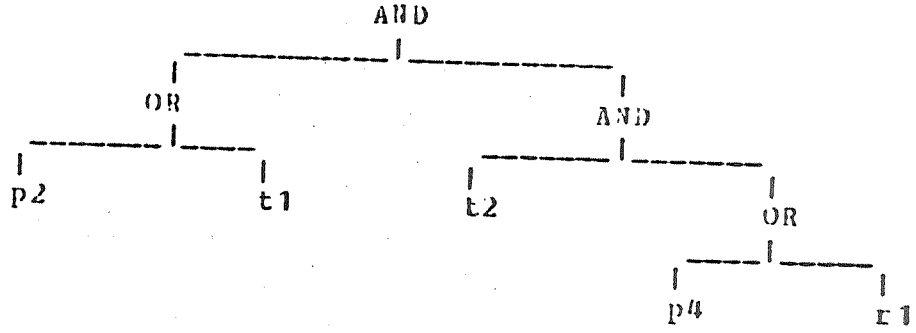


Fig.2 An example of index predicates eligible for multi-cluster queryresolution

It should be pointed out that several queries from several users may be executed in parallel. This raises the problem of scheduling mounts so that the number of mounts is minimized. This problem, however, will not be addressed in the present paper.

7.0 SINGLE-CLUSTER QUERY OPTIMIZATION AND EXECUTION

During the phase of single-cluster query optimization an optimized query strategy execution is generated for a specific cluster. The input to this phase consists of:

1. A set of logical document identifiers denoted by {LDI}. These LDIs are of documents that have been selected by evaluating index predicates in the multi-cluster query resolution phase.

2. A reduced query denoted by RQ.

Recall that RQ is obtained from the original query by removing the index predicates that have been evaluated during the multi-cluster query resolution.

To execute a query a certain number of tasks must be performed. In general it is necessary: (1) to evaluate the text predicates; (2) to evaluate the remaining index predicates; (3) to intersect the set of documents obtained by the tasks (1) and (2) with the set {LDI} obtained from the multi-cluster query resolution phase; (4) to evaluate the residual predicates and detect the false drops.

It is important to notice the following fact. The index predicates remaining in the query after query reduction are the ones which appear in disjunctions with either text predicates or residual predicates, else they would have been selected for multi-cluster query resolution. A consequence of this is that given a disjunction of an index predicate and a text predicate, false drop detection must be performed only for documents that are in the resulting set but do not satisfy the index predicate. This allows us to minimize the number of documents accessed for false drop detection.

Another important observation concerns the fact that it is necessary to access the documents to detect the false drops and to evaluate the residual predicates (if any). In general, accessing documents is rather expensive and it is convenient to restrict the set of documents as much as possible. Therefore it is usually convenient to execute tasks (1) and (2), by using signatures and indexes to reduce the set of documents before accessing the documents themselves. However text predicates and index predicates could be also evaluated on the documents, without using the access mechanisms. This is in general convenient when the text predicates and the index predicates are not very restrictive.

The overall execution strategy can be summarized as follows. First text predicates and index predicates are evaluated to restrict the set of documents. The execution strategy for text restrictions and index predicates is determined by the optimizer and will be described in the next section. Then for each document in the resulting set, one determines the text predicates for which false drop detection must be performed and the residual predicates evaluated. This point will be discussed in section 7.2.

In the remainder of the paper, please read X<i> as X with subscript i.

7.1 Query Optimization

7.1.1 BASIC DEFINITIONS AND NOTATIONS

In describing the various strategies we will use the following operations:

ST(TC<1>,....TC<n>) TOTAL SIGNATURE SCANNING
Input: n text conditions
Output: for each text condition TC<i> the set of LDIs of documents that satisfy TC<i>.

SR(TC<1>,....TC<n>|D) RANDOM SIGNATURE SCANNING
Input: n text conditions, a set D of LDIs
Output: for each text condition TC<i> the set of LDIs of documents that satisfy TCi.

I(p) INDEX SCAN
Input: an index predicate
Output: set of LDIs of documents that satisfy p.

U(D<1>,D<2>) UNION
Input: two sets of LDIs
Output: a set union of D<1> and D<2>

INT(D<1>,D<2>) INTERSECTION
Input: two sets of LDIs
Output: a set of LDIs that are both in D<1> and in D<2>

DIFF(D<1>,D<2>) DIFFERENCE
Input: two sets of LDIs
Output: a set of LDIs that are in D<1> but not in D<2>

A graphical representation of the various operations is given in fig.3.

We will also use the following definitions.

Definition 1

Given a text predicate t<i>, we denote by e<i> the string expression list in t<i>.

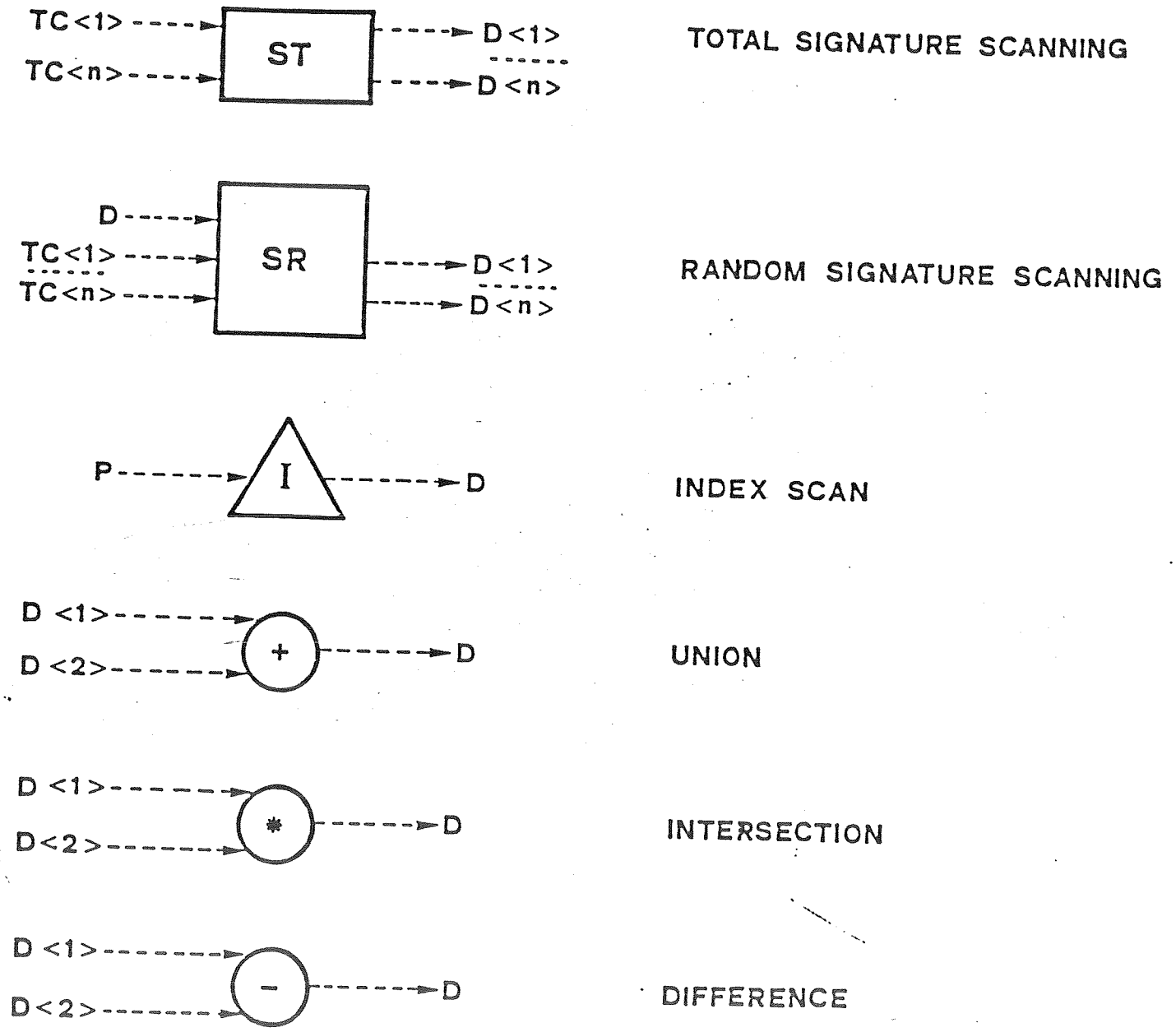


Figure 3 GRAPHICAL REPRESENTATION OF THE OPERATIONS

Definition 2

Given n text predicates t<1>,t<2>,...t<n> (n>1), the concatenation of t<1>,t<2>,...t<n> is defined as the concatenation of e<1>, e<2>,...e<n>. The concatenation will be denoted as e<1>e<2>...e<n>.

Definition 3

Given a query COND clause in conjunctive normal form, we say that a conjunct C<i> is:

a text-conjunct (t-conjunct) if C<i> is a text predicate or a disjunction of text predicates;

a index-text-conjunct (i-t-conjunct) if C<i> is a disjunction of text predicates and index predicates;

a residual-text-conjunct (r-t-conjunct) if C<i> is a disjunction of text predicates and residual predicates;

a residual-index-text-conjunct (r-i-t-conjunct) if C<i> is a disjunction of text predicates,index predicates and residual predicate.

7.1.2 BASIC CASES

A large number of different strategy types can be devised. In this subsection we describe the types of strategies for some basic cases. In the following, D<f> will denote a set of document identifiers obtained from the evaluation of text and index predicates.

7.1.2.1 Case A

Let Q be a query having a COND clause of the form:

C<1> AND C<2> .... AND C<n>

such that each C<i> (i=1,n) is a text predicate t<i>;
let D be a set of documents over which we wish to evaluate Q, then the following strategies can be defined:

Combined Strategy - Type 1 (CST-1)

This type of strategy is based on the principle of evaluating all the text predicates within the query by performing one signature scan. The expression lists associated to text predicates are concatenated to form a single expression list which is passed as argument to the signature handler. This type of strategy has the following steps:

- D<1> = ST([1,e])
  where e = e<1>e<2>....e<n>

- D<f> = INT(D<1>,D)

A graphical representation of this type of strategy is given in fig.4.

Combined Strategy - Type 2 (CST-2)

This type of strategy differs from the previous one in that the a random signature scan is performed for document in D. Therefore there is not need of executing the intersection with D after the signature scan. This strategy has the following steps:

- D<f> = SR({1,e}|D)
  where e = e<1>e<2>....e<n>

A graphical representation of this type of strategy is given in fig.5.

Serial Strategy (SST)

This type of strategy is based on the principle of evaluating serially the various text predicates. This strategy requires the determination of the best order in which to evaluate the text predicates. The steps are as follows:

- D<1> = SR({1,e<i1>}| D)
- D<2> = SR({1,e<i2>}| D<1>)
- .....
- D<f> = SR({1,e<in>| D<n-1>)

where the indexes i1 ,i2 .....,in represent a permutation of the indexes 1,2,...n. A graphical representation of this type of strategy is given in fig.6.

However serial strategies for these types of queries are never convenient. The following assertions hold.

Assertion A.1.

Let Q be a query having a COND clause of the form:

C<1> AND C<2> .... AND C<n>

such that each C<i> (i=1,n) is a text predicate;
let D be the set of documents on which the query is to be evaluated, then:

cost(CST-2) <= cost(SST) always.

Proof The proof is given in Appendix C.

The previous assertion states that the combined strategy - type 2 is always more efficient than the serial strategies.

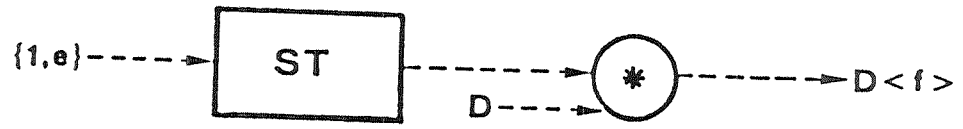


Figure 4 GRAPHICAL REPRESENTATION OF STRATEGY CST-1 (CASE A)

Figure 5 GRAPHICAL REPRESENTATION OF STRATEGY CST-2 (CASE A)

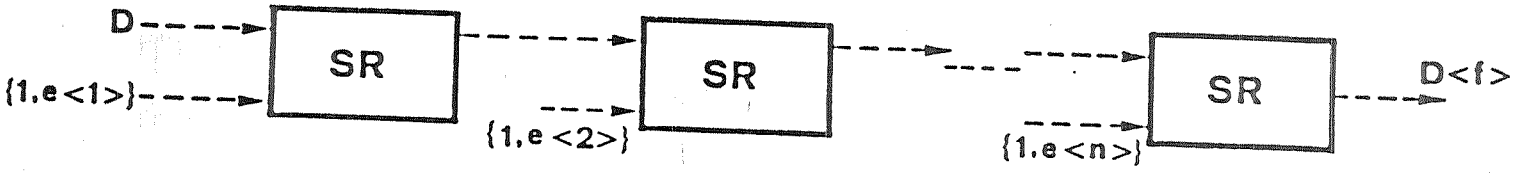


Figure 6 GRAPHICAL REPRESENTATION OF STRATEGY SST-2 (CASE A)

From the previous assertion it can be concluded that for these types of queries the optimization algorithm is very simple since the optimizer must choose only between the two types of combined strategies. In fact, if the cost of the sequential signature scan for card(D) (where D is the set of documents on which the query is to be evaluated) is lower than the cost of random signature access for card(D) then the combined strategy-type 1 is better than the combined strategy-type 2.

7.1.2.2 Case B

Let Q be a query having a COND clause of the form:

C<1> AND C<2> ...AND C<n>

such that each C<i> has the form:

p<i<1>> OR..... p<i<m(i)>> OR t<i<1>> OR .... t<i<s(i)>>

where:

m(i)>=1 and s(i)>=1

each p<i<j>> j=1,m(i) i=1,n is an index predicate

each t<i<h>> h=1,s(i) i=1,n is a text predicate

(Each C<i> is an i-t-conjunct)

let D be a set of documents, then the following strategies can be defined:

Combined Strategy - Type 1 (CST-1)

In this type of strategy the text predicates are evaluated by performing a single signature scan. In this case the signature handler receives as input a number n of text conditions where n is the number of conjuncts in the query. The text condition for a conjunct C<i> contains a number of set expression lists equal to the number of text predicates in C<i>. This type of strategy has the following steps:

- ST(TC<1>,TC<2>...TC<n>);

  where: TC<1> = {s(1), (e<1<1>>, e<1<2>>,....e<1<s(1)>>)}
  TC<2> = {s(2), (e<2<1>>, e<2<2>>,....e<2<s(2)>>)}
  .....
  TC<n> = {s(n), (e<n<1>>, e<n<2>>,....e<n<s(n)>>)}
  let D<1>, D<2>, ...D<n> be the resulting document sets satisfying respectively TC<1>,TC<2>,...TC<n>;

- D'<1> =U(I(p<1<1>>,U(p<1<2>>,..... U(p(<1<m-1(1)>>,p(<1<m(1)>>)

- D'<2> =U(I(p<2<1>>,U(p<2<2>>,..... U(p(<2<m-1(2)>>,p(<1<m(2)>>)

- .....

- D'<n> =U(I(p<n<1>>,U(p<n<2>>,..... U(p(<n<m-1(n)>>,p(<n<m(n)>>)

- D"<1> =U(D<1>,D'<1>)
  D"<1> is the set of documents satisfying the conjunct C<1>

- ....

- D"<n> = U (D<n>,D'<n>)
  D"<n> is the set of documents satisfying the conjunct C<n>

- D<f> = INT(D,INT(D"<1>,INT(D"<2>,.....INT(D"<n-1>, D"<n>))..)

A graphical representation of the strategy for n=2 is given in fig.7.

Combined Strategy - Type 2 (CST-2)

This type of strategy is similar to the previous one except that the signature scanning is executed only for documents in the set D, that is a random signature scanning is performed.

Serial Strategy (SST)

These types of strategies are based on the principle of evaluating serially the various text predicates. This strategy requires determination of is the best order in which evaluate the various conjuncts. Let's suppose that i1,i2,...,in represent a permutation of the indexes 1,2,..,n, then the serial strategy has the following steps:

- D(p<i1>) =
  U (I(p<i1<1>>,U (p<i1<2>>,..... U(p(<i1<m-1(1)>>,p(<i1<m(i1)>>)

- D<i1> = DIFF (D, D(p<i1>))

- D'<i1> = SR({TC<i1>|D(p<i1>))
  where: TC<i1> = {s(i1), (e<i1<1>>, e<i1<2>>,....e<i1<s(i1)>)}

- D"<i1> = U (D'<i1>,INT(D(p<i1>),D))
  D"<i1> is the set of documents satisfying the conjunct C<i1>

- .....

- D(p<in>) =
  U (I(p<in<1>>,U (p<in<2>>,..... U(p(<in<m-1(1)>>,p(<in<m(in)>>)

- D<in> = DIFF (D"<i(n-1)>, D(p<in>))

- D'<in> = SR({TC<in>}|D(p<in>))
  where: TC<in> = {s(in), (e<in<1>>, e<in<2>>,....e<in<s(in)>)}

- D<f> = U (D'<in>,INT(D(p<in>),D<i(n-1)>))

This strategy is based on the fact that given one or more text predicates which appear in a conjunct it is more convenient to evaluate the text predicates only for documents that do not satisfy

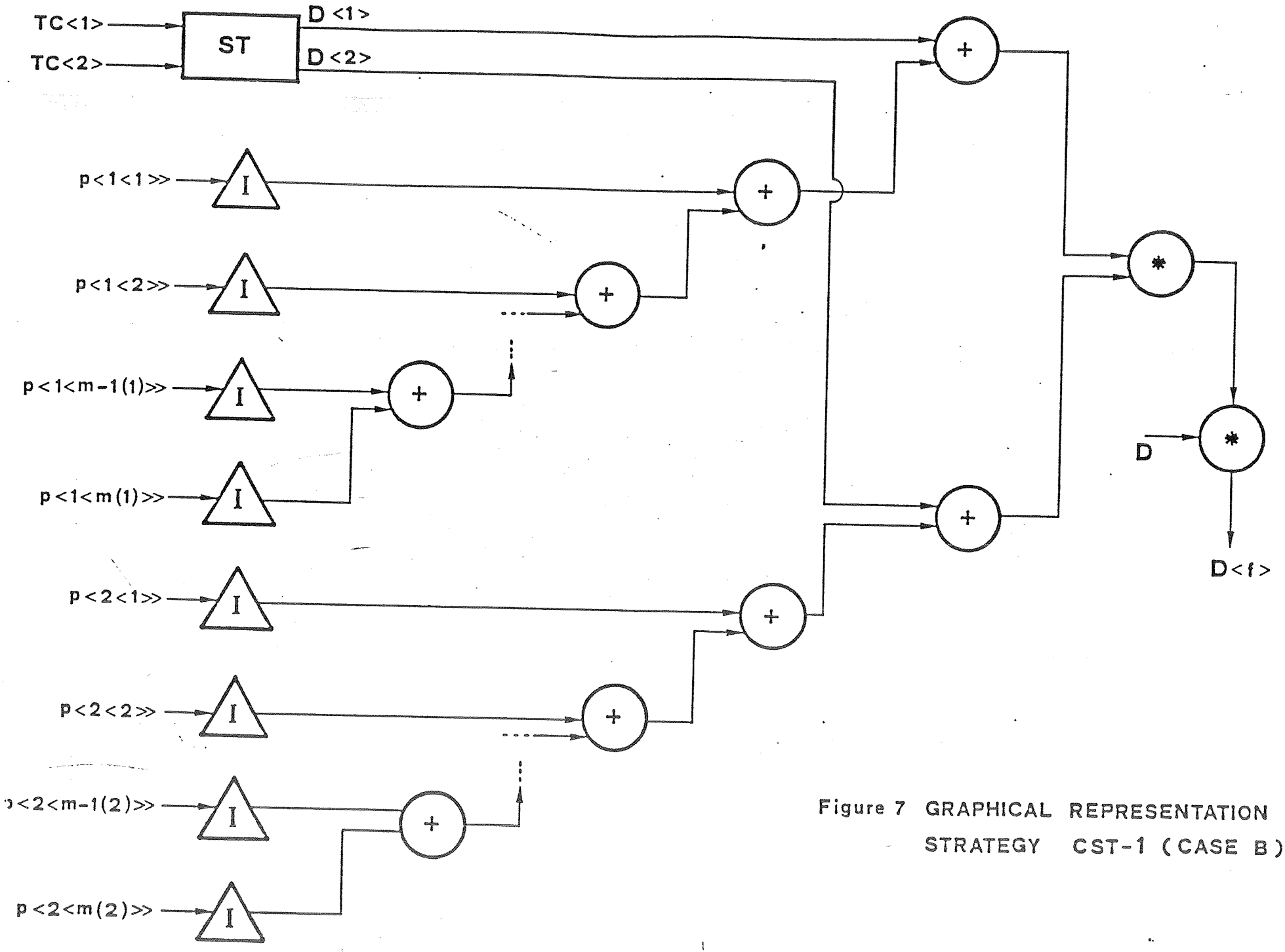


Figure 7 GRAPHICAL REPRESENTATION OF STRATEGY CST-1 (CASE B)

the index predicates that appear in the same conjunct as the text predicates.

A graphical representation of the strategy for n=2 is given in fig.8.

To evaluate the best evaluation order for the various conjuncts the following algorithm is used.

Algorithm B.1.

Step 1

For each conjunct C<i> evaluate:

- s(T<i>) where T<i> is the predicate t<i<1>> OR ...OR t<i<s(i)>>
- s(P<i>) where P(i) is the predicate p<i<1>> OR ...OR p<i<m(i)>>
- q<i> = s(T<i>) * (1-s(P<i>))

Step 2

Order the q<i>'s in increasing order:

q<i1>,q<i2>,....q<in> .

The obtained permutation i1,i2,...in is the output of the algorithm.

For these types of queries, the optimizer has to find the best execution order for the serial strategy by using Algorithm B.1.. Then the cost of the serial strategy is compared with the costs of the combined strategies to find the optimal strategy.

7.1.2.3 Case C

Let Q be a query having a COND clause of the form:

C<1> AND C<2> such that:

C<1> has the form t<1> OR p<1> ;
where t<1> is a text predicate and p<1> an index predicate;

C<2> is a text predicate t<2>;

let D be a set of documents, then the following strategies can be defined:

Combined Strategy - Type 1

This type of strategy consists of executing only one signature scan and evaluating both t<1> and t<2>. The steps are as follows:

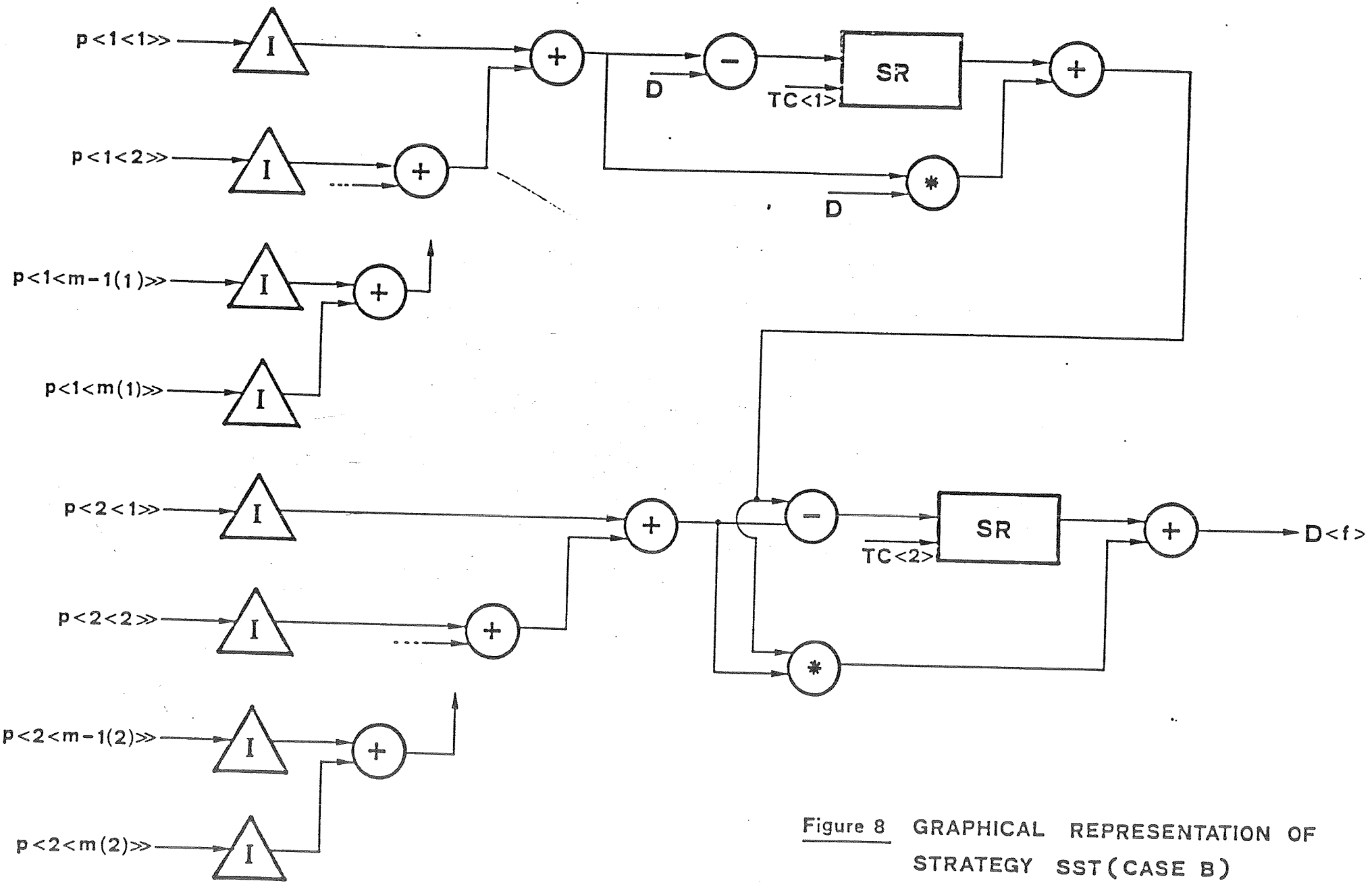


Figure 8 GRAPHICAL REPRESENTATION OF STRATEGY SST(CASE B)

- ST({1,e<1>},{1,e<2>})
  let D<1> and D<2> be the sets of documents satisfying respectively t<1> and t<2>;

- D<3> = I(p<1>) (evaluate predicate p<1> using the index)

- D<4> = U(D<1>,D<3>) (merge D<1> and D<3>);

- D<f> = INT(INT (D<2>,D<4>), D)

A graphical representation of the strategy is given in fig.9.

Combined Strategy - Type 2

This type of strategy is similar to the previous except that the signature scan is restricted to the set of document in D. This type of strategy has the following steps:

- SR({1,e<1>}, {1,e<2>} | D)
  let D<1> and D<2> be the sets of documents satisfying respectively the text conditions {1,e<1>} and {1,e<2>}.

- D<3> = I(p<1>) (evaluate predicate p<1> using the index)

- D<4> = U (D<1>, D<3>)

- D<f> = INT (D<2>, D<4>)

A graphical representation of the strategy is given in fig.10.

Serial Strategy - Type 1

In this type of strategy two separate signature scans are executed to evaluate the text predicates. This type of strategy has the following steps:

- D<1> = SR({1,e<1>} |D)

- D<2> = I(p<1>)

- D<3> = U (D<1>, D<2>)

- D<f> = SR ({1,e<2>} |D<3>)
  in this case the signature scanning for evaluating t<2> is restricted to the set of documents that have satisfied the other conjunct.

A graphical representation of the strategy is given in fig.11.

From this strategy another strategy can be derived where a total signature scan is executed to evaluate t1 and then the intersection with the set D is performed.

Serial Strategy - Type 2

This type of strategy is similar to the previous one, except that the signature scanning to evaluate t1 is executed only for the

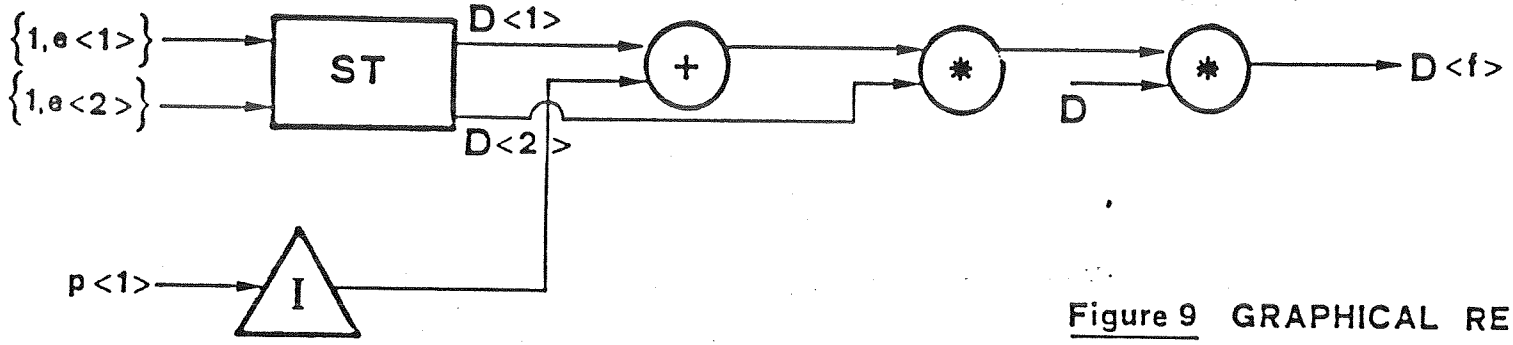


Figure 9 GRAPHICAL REPRESENTATION OF STRATEGY CST-1 (CASE C)

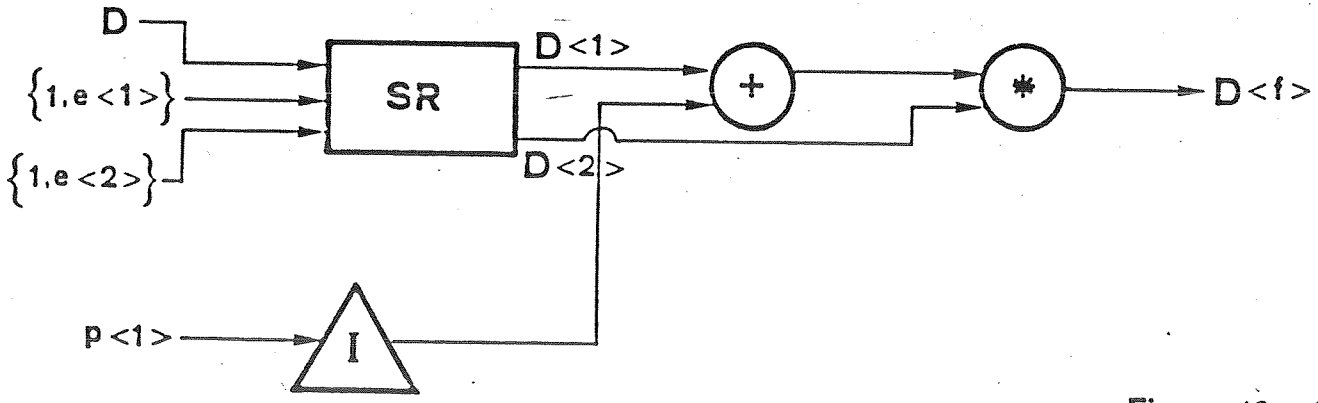


Figure 10 GRAPHICAL REPRENTATION OF STRATEGY CST-2 (CASE C)

documents that do not satisfy p1. This type of strategy has the following steps:

- D<1> = I(p<1>)
- D<2> = DIFF (D<1> ,D)
- D<3> = SR({1,e<1>} | D<2>)
- D<4> = U (D<1>, D<3>)
- D<f> = SR({1,e<2>} |D<4>)

In general this strategy is convenient with respect to the Serial - Type 1 when the predicate p1 is not much restrictive.

A graphical representation of the strategy is given in fig.12.

Serial Strategy - Type 3

This type of strategy is based on the observation that a condition clause of the form:
(t<1> OR p<1>) AND t<2> can be transformed as follows:
(p<1> AND t<2>) OR (t<1> AND t<2>).

In this case the query would be executed evaluating p<1> first, restricting the signature scan for t<2> to the set of documents satisfying p<1> and then performing the signature scan for (t<1> AND t<2>) on only the documents that do not satisfy the disjunct (p<1> AND t<2>). The steps are as follows:

- D<1> = I(p<1>)
- D<2> = INT(D<1>,D);
- D<3> = SR ({1,e<2>} | D<2>)
- D<4> = DIFF(D,D<3>);
- D<5> = SR ({1,e} | D<4>)
  where e = e<1>e<2>
- D<f> = D<3> U D<5>.

A graphical representation of the strategy is given in fig.13.

From the previous discussion it can be seen that several strategies can be defined. However the following assertion holds.

Assertion C.1

Let Q be a query having a COND clause of the form

C<1> AND C<2> where

C<1> has the form t<1> OR p<1>;
C<2> is a text predicate t<2>;

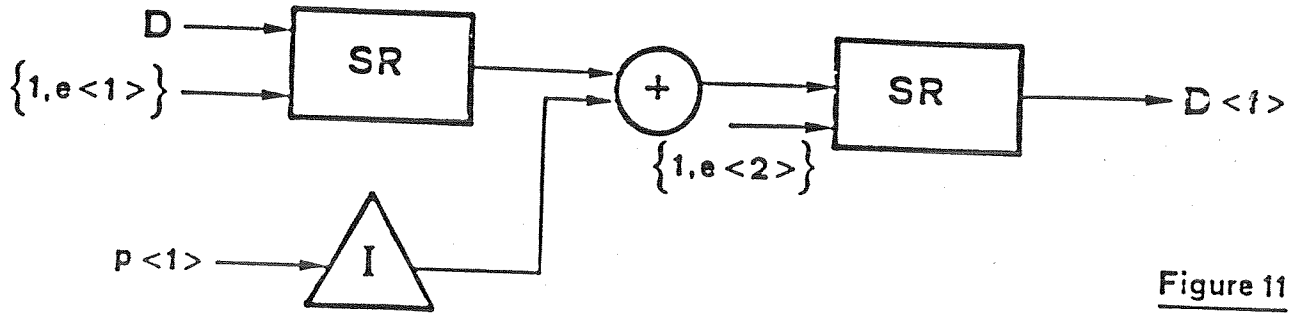


Figure 11 GRAPHICAL REPRESENTATION OF STRATEGY SST-1 (CASE C)

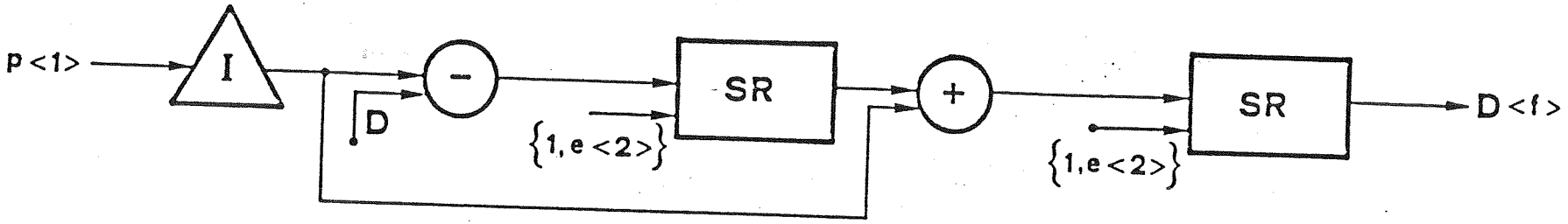


Figure 12 GRAPHICAL REPRESENTATION OF STRATEGY SST-2 (CASE C)

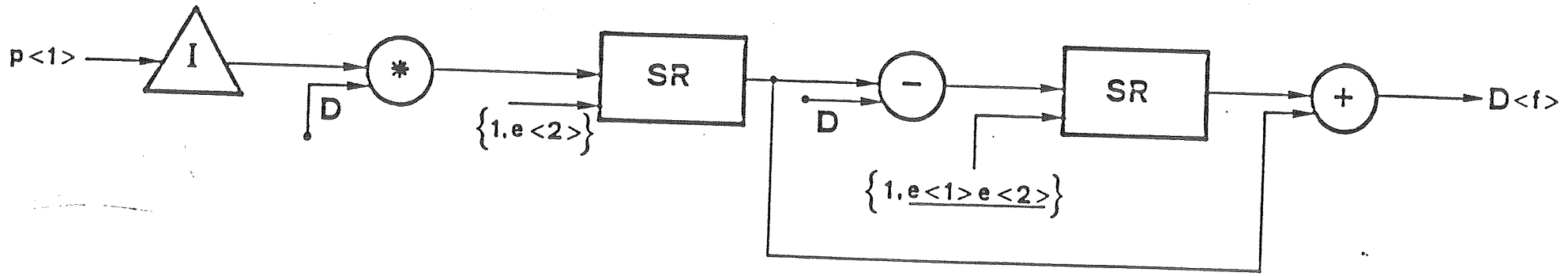


FIGURE 13 GRAPHICAL REPRESENTATION OF STRATEGY SST-3 (CASE C)

let D be the set of documents on which the query is to be evaluated, then:

cost(CST-2) <= cost(SST) always.

Proof The proof is given in Appendix C.

As in case of queries of Type A, the combined strategies are better than the serial ones. Therefore the optimizer has only to decide between a total signature scan and a random signature scan.

We now extend now this result to the more general case.

Assertion C.2

Let Q be a query having a COND clause of the form

C<1> AND C<2> .....C<n> where

a set J (card(J) >=1) of indexes exists such that :

- ∀j ∈ J, C<j> is a text predicate t<j> or a disjunction of text predicates t<j<1>>,t<j<2>>,....t<j<s(j)>> (C<j> is a t-conjunct)

- ∀i ∉ J (i ∈{1,...n}), C<i> has the form
  p<i<1>> OR..... p<i<m(i)>> OR t<i<1>> OR .... t<i<s(i)>>
  where:

  m(i)>=1 and s(i)>=1

  each p<i<j>> j=1,m(i) i=1,n is an index predicate

  each t<i<h>> h=1,m(i) i=1,n is a text predicate

  (C<i> is an i-t-conjunct)

let D be the set of documents on which the query is to be evaluated,

then the combined strategy-type 2 is always more efficient than any serial strategy.

Proof The proof is given in Appendix C.

This assertion states that when the condition clause of a query contains one or more t-conjuncts then the combined strategies are more efficient than the serial strategies.

7.1.2.4 Residual Predicates

If some of the conjuncts are r-conjuncts, or r-t-conjuncts, or r-i-t-conjuncts, then the same strategies described in the previous subsections apply. For the choice of the type of strategy, combined vs serial, a r-t-conjunct is considered as a t-conjunct, while a

r-i-t-conjunct is considered as a i-t-conjunct. For each r-t-conjunct or r-i-t-conjunct a separate text condition is generated for the text predicates of that conjunct. For combined strategies of Case A this text condition is not concatenated with the text predicates of the other conjuncts. Still only one signature scan is performed, the only difference being that instead of one text condition, there will be m+1 text conditions where m is the number of r-t-conjuncts in the query. A residual predicate will then be evaluated on the set of documents that do not satisfy text predicates and index predicates in the same conjunct.

7.1.3 QUERY OPTIMIZATION ALGORITHM

From the previous results, it can be seen that the complexity of query optimization is greatly reduced. The overall query optimization algorithm, presented in fig.14. is decomposed in two main phases.
In the first phase (steps 1,2,3) the most efficient strategy for performing signature scanning is determined. In general the combined strategies are the best, except for queries where all the conjuncts contains index predicates. For these types of queries the optimizer uses the Algorithm B.1. to determine the most efficient serial strategy and then compares it with the cost of the most convenient between the two types of combined strategies.

In the second phase (step 4, 5, 6, 7) the optimizer tries to improve the strategy obtained at the previous step by evaluating strategies where some of the conjuncts are not solved using the access mechanism (indexes and signature) but on the documents themselves. In the fig.14 this set of conjuncts is denoted as RCONJ. It should be noticed that the conjuncts that must be examined are only i-t-conjuncts and r-i-t-conjuncts. The t-conjuncts or r-t-conjuncts instead do not need to be examined. In fact if text predicates in one of such conjuncts are not solved by using the signature mechanism, the cost of signature scan does not decrease while the resulting selectivity is lower and then the number of documents to be accessed increases. Instead for conjuncts containing index predicates some saving can be achieved because the index access is not performed. The set of candidate conjuncts is denoted as ECONJ.

The algorithm examines every conjunct in ECONJ to see if the cost of the query execution decreases by evaluating the conjunct on the documents rather than using the access mechanisms. The conjuncts that increase the cost are eliminated from the set ECONJ (step 5). Then among the conjuncts remaining in ECONJ the one which yields the minimum cost is eliminated from ECONJ and added to RCONJ (step 7). Then the algorithm goes again to step 5. It stops when ECONJ is empty.

1. Let Q be a query, let COND be the condition clause of Q of the form:
C<1> AND .....C<n>
let D the set of documents on which the query is to be evaluated. Let T be the resulting strategy type. Determine the most efficient combined strategy: if the cost of the sequential signature scan for card(D) is lower than the cost of random signature access for card(D) then then T=combined-type-1 (total scan) else T=combined-type-2 (random scan).

2. If an index i, (i=1,n) exists such that C<i> is either a t-conjunct or a r-t-conjunct then goto step (4) else goto step (3).

3. Determine the most efficient serial strategy by using Algorithm B.1. Let I be the index permutation representing the execution order of the various conjuncts.
If cost(serial(I)) < cost(T) then T=serial(I).

4. Determine the set RCONJ of conjuncts that must be evaluated on the documents.
Let Cost0 be the cost of strategy T; let ECONJ denote the set of conjuncts to be examined.
ECONJ = { i/ C<i> is a i-t-conjunct or a r-i-t-conjunct }
RCONJ = 0 If ECONJ = 0 then exit.

5. For each i in ECONJ:

   - evaluate the cost of the strategy obtained from T not evaluating the conjunct C<i> with the access mechanism. Let denote this cost as cost(T(C<i>));

   - If cost(T(C<i>)) > Cost0 then eliminate C<i> from ECONJ.

6. If ECONJ = 0 then exit.

7. Let j ∈ECONJ an index such that:
cost(T(C<j>)) = min { cost(T(C<i>)) / i∈ECONJ} then

   - set T = T(C<j>);

   - set Cost0 = cost(T(C<j>));

   - add j to RCONJ;

   - eliminate j from ECONJ.

   - goto step 5.

Fig.14 Query Optimization Algorithm

7.2 False Drop Detection

False drop detection is performed after signature scans and index accesses. It is performed by retrieving the document text components which are referenced in the text predicates and executing a full text scanning on these components. Since this operation is rather expensive, for each i-t-conjunct (or r-i-t-conjunct) a list of LDIs is generated, after the evaluation of the conjunct, containing the identifiers of documents satisfying text predicates in the conjunct but not the index predicates. These lists are intersected with the final set of documents. Therefore the false drop detection for text predicates in a conjunct C<i> is performed only for documents in the list associated to C<i> that belong to the final set of documents.

8.0 CONCLUSIONS

In this paper we have described query processing in a multimedia document system. Documents can be retrieved by specifying conditions on both document attributes and content. The query language defined is based on a document conceptual model. This means that users can tailor their queries on the base of the document types commonly found in their world. Thereby providing the flexibility needed in environments such the office one.

Document storage is supported by a storage subsystem that integrates both optical and magnetic devices. Thus the system is able to provide the storage capacity needed to store multimedia information. The storage subsystem provides access paths such as indexes, used for search on formatted components of documents, and signatures, used for text searching. In addition it provides bulk store for documents and a table handler, to support the management of various internal data structures and system tables.

The problem of query optimization in such a system has been discussed in detail. First the statistics and selectivity formulae are presented. In addition to the statistics commonly found in DBMSs, statistics for texts are also supported. Then the various steps in query processing have been described, focusing on the query optimization. Despite the large number of possible execution strategies, the results presented in the paper show that for most types of queries the number of possibilities is greatly reduced. To validate our results a simulator [GIBB86] has been built to estimate the execution costs for various strategies.

An implementation of the query processor is underway as part of the ESPRIT (European Strategic Programme for Research in Information Technology) Project 28, called MULTOS. It aims at the realization of a prototype server for multimedia document filing and retrieval based on an open architecture [BERT86]. A first prototype of this server, including the query processor module based on data and text components of documents, is expected by March 1987. A second prototype will follow, in which also image and audio componets will be considered also in the query process [RABI84a]. Future work on the query processing includes the extension of the query optimization to the case of different signature mechanisms, such as bit-sliced organization [ROBE79] or S-tree [DEPP86].

APPEND. A - THE QUERY LANGUAGE GRAMMAR

(Described using the Unix LEX/YACC style)

LEX definitions

```
DIGIT [0-9]
EXPONENT [Ee][+-]?{DIGIT}LETTER [a-zA-Z]
LETTDIGIT [a-zA-Z0-9_]

%%

[  t n]                    ;

[;,!:\(\)\[\]\*\/\.]       {
                    return(yytext[0]);      }

find|FIND                  {
                    return(FIND);   }

version|VERSION            {
                    return(VERSION);        }

first|FIRST                {
                    return(FIRST);          }

last|LAST                  {
                    return(LAST);           }

all|ALL                    {
                    return(ALL);            }

scope|SCOPE                {
                    return(SCOPE);          }

collection|COLLECTION      {
                    return(COLLECTION);     }

type|TYPE                  {
                    return(TYPE);           }

where|WHERE                {
                    return(WHERE);          }

not|NOT                    {
                    return(NOT);            }

and|AND                    {
                    return(AND);            }

or|OR                      {
                    return(OR);             }

with|WITH                  {
```

```
                    return(WITH);           }
between|BETWEEN     {
                    return(OPER_BE);        }
like|LIKE           {
                    return(LIKE);           }
contains|CONTAINS   {
                    return(CONTAINS);       }
is|IS               {
                    return(IS);             }
in|IN               {
                    return(IN);             }
every|EVERY         {
                    return(EVERY);          }
some|SOME           {
                    return(SOME);           }
"="                 {
                    return(OPER_EQ);        }
"!="                {
                    return(OPER_NE);        }
"<"                 {
                    return(OPER_LT);        }
"<="                {
                    return(OPER_LE);        }
">"                 {
                    return(OPER_GT);        }
">="                {
                    return(OPER_GE);        }
{LETTER}+{LETTDIGIT}*   {
                    ...
                    return(IDENTIFIER);     }
{DIGIT}+            {
                    ...
                    return(INTEGER);        }
{DIGIT}+"."{DIGIT}*({EXPONENT})?    |
"."{DIGIT}+({EXPONENT})?            |
{DIGIT}+{EXPONENT}  {
                    ...
                    return(REAL);           }
\"[a-zA-Z0-9_$ ,;:!\?\/\.\-\+]+\"   {
                    ...
```

```
        return(STRING);          }
%%
```

YACC definitions

```
%token FIND
%token VERSION
%token FIRST
%token LAST
%token ALL
%token SCOPE
%token COLLECTION
%token IDENTIFIER
%token TYPE
%token WHERE
%token NOT
%token AND
%token OR
%token WITH
%token OPER_EQ
%token OPER_NE
%token OPER_NE
%token OPER_LT
%token OPER_LE
%token OPER_GT
%token OPER_GE
%token OPER_BE
%token CONTAINS
%token LIKE
%token IS
%token IN
%token SOME
%token EVERY
%token INTEGER
%token REAL
%token STRING

%start query
%left OR
%left AND
%left NOT
%left '*'
%left '.'

%%

query           : find version scope type where_condition
                ;

find            : FIND
                | /* null */
                ;

version         : VERSION FIRST ';'
                | VERSION LAST ';'
                | VERSION ALL ';'
                | /* null */
                ;
```

```
scope               : SCOPE COLLECTION collection_id_list ';'
                    | /* null */
                    ;

collection_id_list
                    : collection_id
                    | collection_id_list ',' collection_id
                    ;

collection_id       : IDENTIFIER
                    ;

type                : TYPE type_id ';'
                    | /* null */
                    ;

type_id             : IDENTIFIER
                    ;

where_condition     : WHERE condition ';'
                    ;

condition           : '(' condition ')'
                    | NOT condition
                    | condition AND condition
                    | condition OR condition
                    | singval_condition
                    | multival_condition
                    | text_condition
                    | with_condition
                    ;

singval_condition
                    : component numeric_predicate
                    | component string_predicate
                    | component date_predicate
                    | component time_predicate
                    | component component_predicate
                    ;

multival_condition
                    : quantifier component numeric_predicate
                    | quantifier component string_predicate
                    | quantifier component date_predicate
                    | quantifier component time_predicate
                    | quantifier conponent component_predicate
                    | component IS IN set
                    ;

text_condition      : component CONTAINS string_list '[' distance ']'
                    | component CONTAINS string_list
                    ;

with_condition      : WITH component
                    ;
```

```
component       : component_half
                | '!' component_half
                | component_half '!' component_half
                ;

component_half  : '*' component_list
                | component_list
                ;

component_list  : component_list '*' component_list
                | component_list '.' component_list
                | component_name
                ;

component_name  : IDENTIFIER
                ;

numeric_predicate
                : operator numeric_value
                | OPER_BE numeric_value numeric_value
                ;

string_predicate
                : operator string_value
                | LIKE string_value
                ;

date_predicate  : operator date_value
                | OPER_BE date_value date_value
                ;

time_predicate  : operator time_value
                | OPER_BE time_value time_value
                ;

component_predicate
                : operator component
                ;


operator        : OPER_EQ
                | OPER_NE
                | OPER_LT
                | OPER_LE
                | OPER_GT
                | OPER_GE
                ;

quantifier      : EVERY
                | SOME
                ;

set             : numeric_list
                | string_list
                | date_list
                | time_list
                ;
```

```
numeric_list    : numeric_value
                | numeric_list ',' numeric_value
                ;

string_list     : string_value
                | string_list ',' string_value
                ;

date_list       : date_value
                | date_list ',' date_value
                ;

time_list       : time_value
                | time_list ',' time_value
                ;

numeric_value   : INTEGER
                | REAL
                ;

string_value    : STRING
                ;

date_value      : '/' INTEGER '/' INTEGER '/' INTEGER '/'
                ;

time_value      : ':' INTEGER ':' INTEGER ':' INTEGER ':'
                | ':' INTEGER ':' INTEGER ':'
                ;

distance        : INTEGER
                ;

%%
```

APPEND. B - SELECTIVITY OF PREDICATES ON ATTRIBUTES

The selectivity of a predicate on an attribute is evaluated as follows depending on the type of restriction [SELI79]:

- comp = value st(p) = 1/NA(comp)

- comp > value

  if comp is a numeric attribute then the selectivity is computed by mean of linear interpolation

  st(p) = (HV(comp) - value)) / (HV(comp) - LV(comp))
  if LV < value < HV

  st(p) = 1 if value <= LV

  st(p) = 0 if value >= HV

  if comp is any other type of attribute then

  st(p) = 1/3

  There is really no significance in this number, except that it is less that 1/2. In fact, as pointed out in [SELI79], it can be expected that few queries use predicates that are satisfied by more than half the tuples.

- comp < value

  if comp is a numeric attribute then:

  st(p) = (value - LV(comp)) / (HV(comp) - LV(comp))
  if LV < value < HV

  st(p) = 1 if value >= HV

  st(p) = 0 if value <= LV

  if comp is any other type of attribute then:

  st(p) = 1/3

- comp BETWEEN (value1, value2)

  if comp is a numeric attribute then:

  st(p) = (value1 - value2)) / (HV(comp) - LV(comp))
  if LV < value1 < HV and
  LV < value2 < HV

  st(p) = (HV(comp) - value2)) / (HV(comp) - LV(comp))
  if value1 > HV

st(p) = (value1-LV(comp)))/(HV(comp)-LV(comp))
if value2< LV

st(p) = 1 if value1>=HV and
value2<=LV

st(p) = 0 if value2<=LV or
value1>=HV

if comp is any other type of attribute then:

st(p) = 1/4

Again there is no significance in this number except that is more selective of a range predicate.

- st(p1 OR p2) = st(p1) + st(p2) - st(p1)*st(p2)
- st(p1 AND p2) = st(p1)*st(p2)
- st(NOT (p)) = 1-st(p)

APPEND. C - PROOFS OF ASSERTIONS IN SECTION 7.

In what follows we denote by:

SK seek time; SK=OSK if the storage is optical else SK=MSK;

TR transfer time; TR=OTR if the storage is optical else TR=MTR;

ALFA it is a constant for given cluster;
ALFA=(NDB(S)*|F/8|)/(NTS*OTR) if the storage is optical else
ALFA=(NDB(S)*|F/8|)/(NTS*MTR) if the storage is magnetic.

In addition we assume that the cost of set intersection, union, and difference is negligible compared to I/O costs.

C.1 Proof of Assertion A.1.

The cost associated to the combined strategy type-2 is as follows:

cost(CST-2) = SK*NSK + ALFA*card(D) + CADDR(card(D)) (1)

where NSK=mincard(D),NB

The cost associated to the serial strategies is as follows:

```
cost(SST-2) = SK*NSK + ALFA*card(D) + CADDR(card(D))
            + SK*NSK<1> + ALFA*card(D<1>) + CADDR(card(D<1>))
            + .........
            + SK*NSK<n-1> + ALFA*card(D<n-1>) + CADDR(card(D<n-1>)   (2)
```

where card(D<j>) = |card(D<j-1>*s(C<i(j-1)>)|

(s(C<i(j-1)>)) represents the selectivity of the (j-1)th conjunct)

card(D<0>) = card(D)

and NSK<j> = min{card(D<j>),NB} j=1,....n-1

From the expressions (1) and (2) it can be concluded that cost(CST-2) <= cost (SST) for whatever selectivity of the text restrictions.

C.2 Proof Assertion C.1.

The cost associated to the combined strategy type-2 is as follows:

$$cost(CST\text{-}2) = SK*NSK + ALFA*card(D) + CADDR(card(D)) + Cost(p<1>) \quad (1)$$

where $NSK=min\{card(D), NE\}$

The cost associated to the serial strategy type-1 is as follows:

$$cost(SST\text{-}1) = SK*NSK + ALFA*card(D) + CADDR(card(D)) + cost(p<1>) + SK*NSK<3> + ALFA*card(D<3>) + CADDR(card(D<3>)) \quad (2)$$

where:

$$card(D<3>) = |card(D) * (s(t<1>) + s(p<1>) - s(t<1>)*s(p<1>)|$$

$$NSK<3> = min\{NE, card(D<3>)\}$$

Comparing the expression (2) with expression (1) it can be seen that $cost(SST\text{-}1) > cost(CST\text{-}2)$ always.

The cost associated to the serial strategy type-2 is as follows:

$$cost(SST\text{-}2) = cost(p<1>) + SK*NSK<2> + ALFA*card(D<2>) + CADDR(card(D<2>)) + SK*NSK<4> + ALFA*card(D<4>) + CADDR(card(D<4>)) \quad (3)$$

where:

$$card(D<2>) = |card(D) * (1- s(p<1>))|$$

$$card(D<4>) = |card(D) * (s(t<1>) + s(p<1>) -s(t<1>)*s(p<1>)|$$

$$NSK<2> = min\{NE, card(D<2>)\}$$

$$NSK<4> = min\{NE, card(D<4>)\}$$

Substituting the values for $card(D<2>)$ and $card(D<4>)$ in the expression (3) we obtain the following:

$$cost(SST\text{-}2) = cost(p<1>) + ALFA* card(D)* (1 + s(t<1>)*(1-s(p<1>)) + CADDR(card(D<2>)) + CADDR(card(d<4>)) + SK * (NSK<2> + NSK<4>) \quad (4)$$

It should be pointed out that:

$$card(D<2>) + card(D<4>) = card(D)* (1 + s(t<1>)*(1-s(p<1>)) >= card(D)$$

the previous inequality holds because $s(p<1>) <= 1$.

From expressions (1) and (4) it can be seen that

$$cost(SST\text{-}2) >= cost(CST\text{-}2)$$

in fact:

- ALFA*card(D)*(1+s(t<1>)*(1-s(p<1>)) >= ALFA*card(D)
  (this holds because s(p<1>)<=1)

- CADDR(card(D<2>)) + CADDR(card(D<4>)) >= CADDR(card(D))
  (this holds because (card(D<2>) + card(D<4>)) >= card(D))

- SK * (NSK<2> + NSK<4>) >= SK * NSK
  (this holds because (card(D<2>) + card(D<4>) >= card(D))

The cost associated to the serial strategy type-3 is as follows:

cost(SST-3) = cost(p<1>)
+ SK*NSK<2> + ALFA*card(D<2>) + CADDR(card(D<2>))
+ SK*NSK<4> + ALFA*card(D<4>) + CADDR(card(D<4>)) (5)

where:

card(D<2>) = |card(D) * s(p<1>)|

card(D<4>) = |card(D) * (1 -s(t<2>)*s(p<1>)|

NSK<2> = min{NE, card(D<2>)}

NSK<4> = min{NE, card(D<4>)}

Substituting the values for card(D<2>) and card(D<4>) in the expression (5) we obtain the following:

cost(SST-3) = cost(p<1>)
+ ALFA* card(D)* (1 + s(p<1>)*(1-s(t<2>))
+ CADDR(card(D<2>)) + CADDR(card(d<4>))
+ SK * (NSK<2> + NSK<4>) (6)

It should be pointed out that:

card(D<2>) + card(D<4>) = card(D)* (1 + s(p<1>)*(1-s(t<2>)) >= card(D)
the previous inequality holds because s(t<2>)<=1.

From expressions (1) and (6) it can be seen as in the previous case that

cost(SST-3) >= cost(CST-2).

C.3 Proof of Assertion C.2.

To demonstrate the assertion we observe that there are three basic types of serial strategies.

In the first type, all the t-conjuncts are evaluated on set D with one signature scan, obtaining a set D', then the i-t-conjuncts are evaluated on set D'. For evaluating the i-t-conjuncts one of

the strategies described for queries of Case B can be applied (combined or serial) In this case the total cost of signature scans for the serial strategy is as follows:

total-cost(signature) = cost(signature scan for document in D)
+ cost(signature scans for document in D') (1)

The cost of signature scan for the combined strategy, however, is as follows:

cost(signature) = cost(signature scan for document in D) (2)

Since the costs of index predicate evaluation is equal in both cases, the combined strategy always performs better than the first type of serial strategy.

In the second type of serial strategy, first the i-t-conjuncts are evaluated for documents in set D (by using strategies defined for queries of case B), obtaining a set D', then the t-conjuncts are evaluated on set D' with one signature scan (this because of Assertion A.1). If the strategy used for evaluating the i-t-conjuncts is the combined then the total cost of signature scan is as follows:

total-cost(signature) = cost(signature scan for document in D)
+ cost(signature scan for document in D')

As in the previous case this cost is greater than cost (2).
If the strategy used for evaluating the i-t-conjuncts is the serial one then several scans are performed to evaluate these conjuncts. Let's suppose that i-t-conjuncts are the first m (this does not lead the generality of the demonstration because of the commutative property of the operator AND), then total cost is as follows:

$$\text{total-cost(signature)} = \sum_{j=1,m} \text{cost(signature scan for conjunct C<j>)} + \text{cost(signature scan for document in D')} \quad (3)$$

To show that expression (3) is greater than expression (2), we show that:

$$\sum_{j=1,m} (\text{card}(D<j>)) + \text{card}(D') > \text{card}(D) \quad (i)$$

$$\sum_{j=1,m} (\text{card}(D<j>)) + \text{card}(D') =$$

$$= \text{card}(D) * [\ 1 - s(P<1>) + \sum_{j=1,m} ((1-s(P<j>)) * \prod_{h=2,j} (s(P<h-1>) + s(T<h-1>) - s(P<h-1>) * s(T<h-1>)))$$

$$+ \prod_{j=1,m} (s(P<j>) + s(T<j>) - s(P<j>) * s(T<j>))\ ] =$$

$$= \text{card}(D) * [\ 1 + s(T<1>) * (1-s(P<1>))$$

$$+ \sum_{j=2,m} (s(T<j>) * (1-s(P<j>)) * \prod_{h=2,j} (s(P<h-1>) + s(T<h-1>) - s(P<h-1>) * s(T<h-1>)))$$

$$+ s(T<m>) * (1-s(P<m>)) * \prod_{J=1,(m-1)} (s(P<j>) + s(T<j>) - s(P<j>) * s(T<j>))\ ]$$

Append. C - Proofs of Assertions in Section 7.

$$= card(D) * [1 + BETA] \qquad (4)$$

In the previous expression T<j> is a predicate defined as the OR of all the text predicates in conjunct C<j>. P<j> is defined in the same way as the OR of all the index predicates in conjunct C<j>. BETA is the espression in square brackets and it is a number greater than zero, since it is obtained as sumns and products of numbers greater than zero. Therefore we have that

$$card(D) * [1 + BETA] > card(D)$$

Therefore the (i) is demonstrated.

In the third type of serial strategy, first some of the i-t-conjuncts are evaluated for documents in set D (by using strategies defined for queries of case B), obtaining a set D', then the t-conjuncts are evaluated on set D' with one signature scan, obtaining a set D", then the remaining i-t-conjuncts are evaluated on D". However, from the previous two cases, it can be easily deduced that also in this case, the combined strategy is better than this type of serial.

9.0 REFERENCES

[ASTR76] Astrahan M.M. et Al., "System R: Relational Approach to Database Management", ACM Trans. on Database Systems, Vol.1, N.2, June 1976, pp.97-137.

[ASTR80] Astrahan M., Kim W., Schkolnick M., "Evaluation of the System R Access Path Selection Mechanism", IBM Research Report RJ2797 San Jose (Calif.), April 1980.

[BARB85] Barbic F. and Rabitti F., "The Type Concept in Office Document Retrieval", in Proc. VLDB Conference, Stockholm, August 21-23, 1985.

[BAYE72] Bayer R., McCreight E., "Organization and Maintenance of Large Ordered Indexes", Acta Informatica, Vol.1, 1972, pp.173-189.

[BERT85] Bertino, E., Gibbs, S., Rabitti, F., Thanos, C., Tsichritzis, D., "Architecture of a Multimedia Document Server" Proc. 2nd ESPRIT Technical Week, Brussels, Sept. 1985.

[BERT86] Bertino E., Gibbs S., Rabitti F., Thanos C., and Tsichritzis D., "A Multimedia Document Server", Proc. Advanced Database Symposium, Japan, August 29-30, 1986.

[CARD75] Cardenas, A.F., "Analysis and performance of inverted database structures". Comm. ACM, Vol.18, N.5, (May 1975), pp.253-273.

[CHRI84] Christodoulakis S., "Implications of Certain Assumptions in Database Performance Evaluation", ACM Trans. on Database Systems, Vol.9, N.2, (June 1984), pp.163-186.

[CHRI84a] Christodoulakis, S. and Faloutsos, C., "Design considerations for a message file server," IEEE Trans. on Software Engineering Vol. SE-10(2), pp.201-210 (March 1984).

[CHRI85] Christodoulakis S., "Multimedia Data Base Management: Applications and Problems", position paper, Proc. ACM-SIGMOD Conference, Austin TX, May 28-31, 1985.

[COMP85] Computer, Special Issue on Multimedia Communications, Vol.18, N.10, October 1985.

[DEPP86] Deppisch U., "S-Tree: A Dynamic Balanced Signature Index for Office Retrieval", to appear in Proc. ACM Conference on Information Retrieval, Pisa (Italy), September 8-10, 1986.

[ECMA85] ECMA TC-29, "Office Document Architecture", Standard ECMA-101 (Sept. 1985).

[FALO84] Faloutsos C., Christodoulakis S., "Signature Files: An Access Method for Documents and its Analytical Performance

Evaluation", ACM Trans. on Office Information Systems, Vol.2, N.4, pp.267-288, Oct.1984.

[GEHA82] Gehani, N., "The potential of forms in office automation," IEEE Trans. on Commun. Vol. Com-30(1), pp.120-125 (Jan. 1982).

[GIBB83] Gibbs S., and Tsichritzis D., "A Data Modelling Approach for Office Information Systems", ACM Trans. on Office Information Systems, 1983.

[GIBB86] Gibbs S., "Document Query Simulation and Processing Heuristics", I.E.I.-C.N.R. Working Paper, March 1986.

[HORA85] Horak W., "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization", Computer, Vol.18, N.10, pp.50-62, October 1985.

[IEEE84] IEEE Database Engineering, Special Issue on Multimedia Data Management, Vol.7, N.3, September 1984.

[ISO84] ISO, ISO/TC 97/SC 18/WG 3 N 283, "Office Document Architecture", 1984.

[MURP83] Murphy J., "Integrated Office-Automation Systems", Mini-Micro Systems, Vol.16, N.6, May 1983.

[POST82] Postel J., "Internet Multimedia Mail Document Format", Technical Report RFC 767, DARPA Network Working Group (March 1982).

[RABI84] Rabitti, F. and Zizka, J., "Evaluation of access methods to text documents in office systems," Proc. 3rd Joint ACM-BCS Symposium on Research and Developement in Information Retrieval (1984).

[RABI84a] Rabitti, F., Stanchev, P., "What can we do with Images in a Multi-Media Document Filing System?" Proc. of the AICA-84 Annual Conference, Italy, (1984).

[RABI85] Rabitti F., "A Model for Multimedia Documents", in "Office Automation: Concepts and Tools", D.Tsichritzis ed., Springer-Verlag, 1985.

[RABI86] Rabitti F., "MULTOS Project: Document Model", ESPRIT Project 28, Report DM-[IEI]-86-02, March 1986.

[ROBE79] Roberts C.S.,"Partial-match Retrieval via the Method of Superimposed Codes", Proc. IEEE, Vol.67, N.12, pp.63-69, December 1979.

[SALT83] Salton, G. and McGill, M.J., Introduction to Modern Information Retrieval, McGraw-Hill (1983).

[SELI79] Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., Price T.G., "Access Path Selection in a Relational Database

Management System", IBM Research Report RJ2429, San Jose (Calif.), January 1979.

[SVOB84] Svobodova L., "File Servers for Network-based distributed Systems", ACM Comp. Surveys, Vol.16, N.4, pp.353-398, Dec.1984.

[THOM85] Thomas R.H., et Al., "Diamond: A Multimedia Message System Built on a Distributed Architecture", Computer, Vol.18, N.12, Dec.1985.

[TSIC83] Tsichritizis D., and Christodoulakis S., "Message Files", ACM Trans. on Office Information Syst., Vol. 1(1), pp.99-98 (Jan. 1983).

[WHAN83] Whang, K.Y., Wiederhold, G., Sagalowicz, D., "Estimating Block Access in Database Organizations: A Closed Noniterative Formula", Comm. ACM, Vol.26, N.11, Nov. 1983

[YAO79] Yao B.S., "Optimization of Query Evaluation Algorithms", ACM Trans. on Database Systems, Vol.4, N.2, June 1979, pp.133-155.