A Query Optimization Method for Use in a Generalized Database

Caroline D. Mautz

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/1328
A QUERY OPTIMIZATION METHOD FOR USE
IN A GENERALIZED DATABASE

by

Caroline D. Mautz

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 1986
Currently proposed methods for retrieval of records from a generalized file organization are not optimal, causing more I/O retrievals than are necessary. This paper proposes another technique that has near optimal results and has a polynomial order of complexity.

The technique takes as input a query in disjunctive form, sorts the keys in the query in ascending order according to the number of records associated with them, then using a table much like a prime implicant table, systematically searches for a complete cover of the query conjuncts with a minimum total number of records associated with the keys that are chosen for the cover.

This technique along with two current query optimization techniques were implemented and an a posterior analysis of the algorithm was made for reliability in finding the optimal solution to randomly generated queries.
ACKNOWLEDGMENTS

I would like to thank Professor Dionysios Kountanis for his cheerful help with the progress of this thesis, Professor Dalia Motzkin for her encouragement, Mr. Bob Trenary for his willing help, and my dear sons for their patience with me as I struggled to achieve this goal.

Caroline D. Mautz
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Mautz, Caroline D.

A QUERY OPTIMIZATION METHOD FOR USE IN A GENERALIZED DATABASE

Western Michigan University M.S. 1986

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark □.

1. Glossy photographs or pages □
2. Colored illustrations, paper or print □
3. Photographs with dark background □
4. Illustrations are poor copy □
5. Pages with black marks, not original copy □
6. Print shows through as there is text on both sides of page □
7. Indistinct, broken or small print on several pages □
8. Print exceeds margin requirements □
9. Tightly bound copy with print lost in spine □
10. Computer printout pages with indistinct print □
11. Page(s) _______ lacking when material received, and not available from school or author.
12. Page(s) _______ seem to be missing in numbering only as text follows.
13. Two pages numbered _______. Text follows.
14. Curling and wrinkled pages □
15. Dissertation contains pages with print at a slant, filmed as received □
16. Other □

University Microfilms International

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# TABLE OF CONTENTS

## ACKNOWLEDGEMENTS

## CHAPTER

I. INTRODUCTION TO THE PROBLEM
   - Types of Queries
   - File Structures
   - Record Retrieval

II. REVIEW OF THE LITERATURE

III. CURRENT QUERY OPTIMIZATION TECHNIQUES
   - Introduction to the Problem of Query Optimization
   - An Overview of Query Processing Methods
   - Complexity
   - Greedy and Dynamic Programming Methods
   - The Prime Key Word Technique
   - The Minimum/Maximum Technique

IV. DESCRIPTION OF THE COMBINATION TECHNIQUE

V. TESTING OF THE RETRIEVAL MINIMIZATION TECHNIQUE
   - The Algorithms and Their Reliability
   - Complexity of Retrieval Minimization Techniques

VI. CONCLUSIONS AND RECOMMENDATIONS
   - Recommendations for Further Research

## APPENDICES

A. Random Query Generator and Queries
B. Programs for Testing the Three Algorithms
C. Results

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table of Contents - Continued

BIBLIOGRAPHY............................................................74

iv
CHAPTER I

INTRODUCTION TO THE PROBLEM

One of the most valuable resources of an enterprise is the information that can be extracted from its database. This database, typically, is a computer based record keeping system that records, maintains, and accesses information. This information is anything deemed important to the organization and is data that are significant to the decision making process that keeps the organization running smoothly.

The technology of designing an efficient database is one of the fastest growing areas of computer and information science. There are many levels of interaction with a database, depending on who is using it and a designer must take all of these views into account. One commonly speaks of the conceptual, the logical, and the physical view of a database. One of the key requirements of the conceptual view of a database, which is the user's perception of it, is that it should satisfy the user's need for information in a reasonable time. In other words, it should satisfy the performance requirements of the user. Ways of satisfying this objective include (a) providing an efficient and easy to use query language, (b) supplying an on-line system, (c) creating software that analyses the query and simplifies it so that the retrieval of records from secondary storage is reduced, thus reducing the time that the user must wait before the query is satisfied, and (d) devising a physical storage method that shortens
the disk access time. The logical view, which is closely tied to the
query language and the software for record retrieval, affects the
user's or conceptual view. The physical view, which is comprised of
the actual allocation of records to physical storage, also is a fac-
tor in satisfying the user's performance objectives.

This paper covers one of these aspects. This aspect is that of
analyzing a query and simplifying it so that as few records as possi-
ble are retrieved.

To give a general background of the concepts and terms used in
this paper, a brief review of queries, file structures, and record
retrieval methods is offered.

**Types of Queries**

The type of query dealt with in this paper is a request by a
user for a record or records that have a designated value for one or
more key values. To use the classification system developed by
Cardenas(1973), the simplest query is an atomic condition, A. This
will take the form:

\[
\begin{align*}
\text{Name} &= \text{Value} \lor \\
\text{Name} &= \text{Value} \lor \\
\text{Name} &< \text{Value} \lor \\
\text{Name} &> \text{Value} \lor \\
\text{Name} &\leq \text{Value} \lor \\
\text{Name} &\geq \text{Value}
\end{align*}
\]

where Name is the field name of the key.

An item condition, I is a disjunction of atomic conditions of
the form:

\[ I_1 \text{ AND } I_2 \text{ AND } I_3 \text{ AND } \ldots \text{ AND } I_m \]

where each I designates a different field name. For example:

(Major = CS OR Major = Math) AND (Age = 20 OR Age = 30)

A query condition, \( Q \), is a disjunction of record conditions of the form:

\[ R_1 \text{ OR } R_2 \text{ OR } R_3 \text{ OR } \ldots \text{ OR } R_n \]

For example:

\[((\text{Major} = \text{CS} \text{ OR } \text{Major} = \text{Math}) \text{ AND } (\text{Age} = 20 \text{ OR } \text{Age} = 30)) \text{ OR} \]

\[((\text{Sex} = \text{Male}) \text{ AND } (\text{Degree} = \text{MS}))\]

Almost all queries can be expressed in one of the Boolean forms shown above. There is an additional query condition that has received attention from Tahani (1977). It is called the fuzzy query and it allows a query such as "very tall" to be used. This query is briefly described in the next chapter. This paper does not deal with fuzzy queries or queries that include inequalities. Rather it deals with Boolean queries that are stated in the disjunctive form, a form that all Boolean expressions can be reduced to.

**File Structures**

Hsiao and Harary (1970) described a generalized file structure for random access storage. In the index for this file structure, each keyword, \( K \), has associated with it a number, \( N \), which indicates the number of records in the file that contain keyword, \( K \). These records that contain a common key are linked together in one or several linked lists. Within the index, each keyword, \( K \), will also have...
a number, $h$, that indicates the number of linked lists and a list of beginning addresses for these lists. Thus a key, $K$, with 15 records associated with it in 3 linked lists would have an index entry of: $K,15,3,adr_1,adr_2,adr_3$.

The generalized file structure which is used in this paper encompasses many methods of storage of records. At one extreme, where $h = 1$, is the multilist where every key will have associated with it only one linked list. At the other extreme, when $h$ is equal to $N$, the file structure is an inverted file where every list contains only one record and each address is listed in the index. Other file organizations would have a value of $h$ between 1 and $n$ and could be represented by this generalized file structure.

In order to access a record, an address for that record must be known. A key value within the record could be used as the actual address for the record, if that value is unique. Only one record can be stored at a given address. This direct access method is not practical for most records and would not allow access to the record except by using the attribute specified by that keyword. For example, in a University record containing a Social Security number, a name, and a major, it would be possible to assign each record to the address designated by the Social Security number, because that is unique. To find all students with a given major would be much more difficult, because those records cannot be directly accessed.

A directory could be created in a file that contains an index for each attribute and within the index, all possible values for the attribute and a list of addresses of records that contain that at-
tribute value. A file containing such a directory would be called an inverted file. Every attribute is accessible. To find all records containing a given attribute value, one would search the index of that attribute for the wanted value. Once that is found, the list of addresses can be used to access all needed records. The inverted file can require a great deal of storage space for its directory because every record address is listed in each of the indexes. To avoid this problem, a multilist storage method could be used. Within the indexes for the attribute, for each value that an attribute has in the file, only one address is stored. This address is the address of the header node of a linked list of records that contain that attribute value. Each record would be required to have as many linked list fields as there are indexes. This would cause the directory size to be small and the storage space for records to increase. This is a drawback as well as having a longer time required to sequentially access the linked lists.

A compromise between these two methods which would shorten the access time and still have a directory that can be stored in primary storage could be used. The length of the list could be limited in some way, by specifying a maximum length for the lists or by limiting the list to the contents of a cell which can be a sector, track, or page of memory. This latter method is called a cellular method. Thus only the beginning address of a list or cell would be included in the index entry.
Record Retrieval

The physical model of the database plays an important part in its visible performance. To the user, the response time is the most apparent performance criteria. The time element is one of the areas that a database designer tries to optimize. The other area of concern to a database designer is the space that the database requires in memory or on a disk. Usually there is a trade off between the number of I/O operations and the amount of redundant data that is stored. Some database designs will allow for a quicker retrieval if there is an index to one or more of the keys, but this means that a record containing more than one of these keys will be listed in more than one index. Another consideration that affects the performance of a database is the actual physical arrangement of the records in the database. For example if a record is perceived conceptually as being the next record, but actually is in a different block, a greater time for accessing that next record will be taken than it would be if it actually was physically contiguous. A database could be very inefficient if this aspect is not taken into consideration.

The actual retrieval of a record from a secondary storage device is the most time consuming operation in satisfying a query. This operation takes about a thousand times as long as a machine instruction. For this reason, designing a method that will reduce the number of these retrieval operations has attracted the attention of researchers and is the goal of this paper.

Designers of data bases have implemented methods that reduce the number of these retrievals as well as the efficiency of the retrieval
itself by reducing the steps needed to effect the retrieval. To reduce the number of record retrievals, the trade off is to increase the processing of the query to insure that the minimum number of records that could satisfy the query is being retrieved. This is the approach that this paper takes, rather than addressing the problem of designing a database physically. It is assumed that this matter has been considered and that the database is as physically efficient as can be achieved for the environment in which it exists.

Thus we see that there are many factors that effect the speed with which a query can be satisfied. All levels of the database are important and have an impact on query response.

In succeeding chapters, the following topics will be covered:

Chapter 2 contains a review of literature related to reducing the time it takes to satisfy a query. The different approaches taken in these papers are: reducing the query to a simpler Boolean expression, developing theories on how to choose a database design for a specific set of data, designing efficient hashing methods, and designing different addressing methods for the records. This review is not exhaustive, but includes literature gathered from the last fifteen years of publications.

Chapter 3 presents an introduction to the problem of query optimization along with a discussion of some of the methods used to evaluate optimization techniques and then a discussion of two of the current query processing optimization strategies along with examples showing their shortcomings.

In Chapter 4, the query optimization method that this paper pro-
poses is presented with examples to illustrate the method.

Chapter 5 is a report on the actual use of this proposed query optimization scheme and a comparison of its efficiency with that of the current query optimization schemes. The order of complexity of each scheme is calculated.

Chapter 6 is a summary of this work with suggestions for further research.
CHAPTER II

REVIEW OF THE LITERATURE

Through the years, attention and effort have been focused on the problem of reducing the time that it takes to access records that satisfy a user's query to a database. In the body of research done on this subject, this problem has been approached in several ways. One approach is an analysis of the query itself, reducing it to an equivalent Boolean expression which ultimately will cause fewer records to be retrieved from the database. Design of the database has an impact on the retrieval speed, so formulas and theories to predict the best database design for a given set of data have been developed. Also, different hashing methods have been proposed to improve the speed of database operations. The actual physical addressing of the database records is another topic of interest to those interested in reducing query response time.

This chapter contains a review of literature on topics relating to this attempt to reduce response time of queries and to understand the factors that have a bearing on this problem.

We will first look at the research done in an attempt to find the best conceptual database design for a given set to data.

Lefkovitz (1969) wrote a classic book dealing with file structures. It deals with the fact that in an information system or database, design considerations that relate to speed of system response are made on the basis of the speed of update, of presearch stats-
tics, ease of programming, and list structure overhead in time and space. He presented formulas that aid the design of efficient file structures for retrieval of information. For example, he developed a formula that predicts when an inverted list search is faster than a multilist search. If we know:

- \( L_i \), the average list length,
- \( N_t \), the average number of terms in a single query product,
- \( L_s \), the shortest list length in a query,
- \( L/A \), the number of physical records per average list length in the query,
- \( p \), the ratio of query response to the shortest list length in the query,

when the inequality, \( p > 1 - N_t/L_s \times (L/A) \) is true, Lefkovitz states that an inverted list search is faster than a multilist search.

The problem of determining which lists to access for a complex query made up of sums of products, or in other words, the disjunctive form, is not addressed and neither is an attempt made to find an optimal key combination before the actual processing of keys is done. The key combination in Lefkovitz's work is simply made up of the shortest key list in the key product of each conjunct. In other words, the Prime Key Word method was used.

Hsiao and Harary (1970) is one of the papers upon which many others depend for a definition of a generalized file structure. In their paper, a file structure is developed which encompasses an inverted file on the one hand and a multilist file on the other. The record, \( R \), is defined as a cartesian product of attributes and values.
in which an attribute has only one value, causing it to be in first normal form. An index of a record is defined as a set of attribute value pairs or keywords that characterize the record. The address of a record shows where the record is stored physically. Sometimes a record, \( R \), has associated with one of its keywords, \( K \) an address which indicates another record with the same keyword. This is called a \( K \)-pointer. A \( K \)-list, or a list, \( L \), of records with the same keyword, \( K \), is a set of records in which the following is true: the \( K \) pointers are not duplicated, only records with the same key are pointed to in the list, the only record not pointed to in the list is the record at the beginning of the list, and only one record has a null \( K \)-pointer which designates the end of the list. A set, \( F \), of such records is called a file and a directory to that file would contain information about the keywords as follows: if there are \( m \) different keywords \( K_1, K_2, \ldots, K_m \); \( n_i \) is the number of records containing the keyword, \( K_i \); \( h_i \) is the number of \( K_i \) lists in \( F \); and \( a_{ij} \) is the beginning address of the \( j \)th list, we then have in the directory a sequence containing the following: \( K_1, n_1, h_1, a_{11}, a_{12}, \ldots, a_{1n_1} \) for \( i = 1, 2, \ldots, m \). A generalized file is defined as a file and its directory.

Thus an inverted file is one in which every list contains one record and a multilist file structure, which is the one to which this paper applies, is one in which only the beginning address of each list for each keyword is contained in the directory.

Hsiao (1971) proposes that a template for record organization or reorganization should be worked out by a designer on both a global...
and a local level. By separating these two levels from each other when designing the database, the storage can be made more efficient, records can be more easily reorganized without changing the values and record control block, and more than one way of organizing the records can be implemented if more than one template for the global level is used.

Cardenas (1973) is concerned with evaluating file organizations and then selecting one on the basis of storage costs and access time to answer queries. He again raises the issue of reducing the average time to answer an average query. This and other factors are named as significant criteria for determining the best organization for a given database. Simulation of several database organizations using six real databases was done.

Cardenas (1975) analyzes the inverted database and derives formulas to estimate the average access time. He shows the interaction of database content, the logical complexity of queries, and the machine timing and blocking specifications. The point is made that the directory should be considered a database in and of itself. He presents the opinion that the directory should be viewed as a part of a hierarchy of three levels of inversion rather than as a sequential file. Formulations are presented to be used in conjunction with the index selection criteria to find the optimum set of index keys.

The complexity of a query is considered as a most important input parameter. In the actual testing of various file organizations with queries, he used queries with varied complexity. He used queries that varied from an atomic condition to one in which there were
eight record conditions. An atomic condition is a request for one value of one key and a record condition is a conjunction of disjunctions of atomic conditions. The query response time was heavily dependent on the file organization, with the multilist organization, in over half of the cases, taking less time. The multilist appeared to handle complex queries more gracefully than the inverted file or doubly-chained tree, although when the list length was long, the inverted file was the best choice. No discussion was made as to how the queries were simplified to reduce the number of records retrieved per query in the multilist, which would have had an impact on the time needed to satisfy a query to a multilist.

The physical organization of the database along with several ways of physically addressing records has drawn attention with various solutions proposed to reduce query response time given by researchers.

Rothnie and Lozano (1974) suggests a method called multiple key hashing or mkh to reduce page accessing by using a hashing function that groups records with the same key on a small number of pages, with a way to determine which pages contain these records, without accessing extra pages to retrieve a request. He states that unless the number of retrievals is less than the number of pages containing the file, attempts to reduce retrievals from the file are not very useful. The restructuring of the file that he suggests can reduce the effective size relative to a retrieval request.

The hashing function is chosen relative to the number of records containing a certain keyword, such that the number of pages is re-
duced, yet all of the records containing the keyword are on as few pages as possible.

McDonell (1977) discusses various ways to organize, create, and maintain an inverted index on a random access storage device. Using a simulation model and a hashed addressed random access model, he found that the three parameters that affect performance of an inverted index are: bucket length, which is the number of fragments allowed in a physically contiguous space, fragment length, which is a fixed portion of a bucket, and packing density, which is the fraction of available fragments that are being used.

Chang (1984) notes that many algorithms have been proposed to construct a minimum perfect hashing function, which is a one to one, onto relationship between key spaces and address spaces. The only one, in Chang's opinion, that is not a heuristic is Jaeschke's method that guarantees a minimum perfect hashing function. Chang's method is also claimed to be minimum perfect, as well as being hashed in ascending order.

The problem of determining how many bits each attribute should be mapped by, in a partial match query, which is a query that specifies only some of the attributes of a record, was addressed by Moran (1983). The objective was to minimize the number of buckets retrieved per query. Records are stored in buckets and the bucket in which a given record is stored is found by a multiple key hashing function, which maps each attribute to a string of a fixed number of bits. The address of that bucket is then represented by the string obtained by concatenating the strings on which the attributes were
mapped. A partial match query may specify only part of the bits in the string representation of the address. The more bits specified, the fewer buckets needed to be retrieved. This problem is NP-hard, or in other words, this problem cannot be solved in polynomial time. Two heuristic algorithms were presented.

Several table look-up methods have been proposed to facilitate the satisfying of queries.

Papakonstantinou (1974) proposes a technique based on a "table of satisfiable terms". This technique makes the assumption that each keyword occurs only once per query, the order of keywords in each query is irrelevant, and if a term is satisfiable then all terms composed of keywords belonging to the first term are satisfiable. This technique enables a search for an entry to have a time that is a function of the number of variables in the corresponding term. The assumptions made for this technique do not apply to queries dealt with in this paper, because in this paper the problem of dealing with keywords that occur more than once in a query is a main issue.

Wong and Chiang (1971) in an effort to eliminate the time necessary for taking the intersection of key attribute lists, propose a new file structure made up of "atoms" of the collection derived of all retrievable sets. These atoms are irreducible units of a Boolean algebra derived from the records in the file.

The structure consists of lists of atoms rather than lists of keywords. Each keyword is on only one list and each set to be retrieved is a union of disjoint atoms so there is no need to take an intersection or to have to eliminate duplication when forming an un-
ion. The query is transformed into a union of atoms by expressing it in a developed disjunctive normal form. Because every non-void clause in this form corresponds to an atom, comparing these with the atoms in the table provides a set of addresses that can be retrieved to satisfy the query. This would be useful only for a static database, because any deletions or additions to the records would change the Boolean algebra which is based on the records of the file.

Tahani (1977) discusses an interesting problem in which a query uses imprecise or "fuzzy" predicates. Such a predicate would be a description like "old" or "tall". This is an attempt to let queries represent human thought, which is often imprecise.

If we think of someone as old, we do not have one value, say 60, as a value that defines old and no other age having that definition, rather, we compare a given age to that value and decide how closely it complies with our perception. The fuzzy term is assigned a universe of discourse, or numerical range of values, which tells to what degree a value complies with the fuzzy description. On the basis of this numerical range, records can be chosen that satisfy the fuzzy query.

Looking at the Boolean expression of the query and simplifying it is the topic of some researchers. Some were seeking a solution for a sequential database and others for a random access system.

Hanani (1977) proposes a solution to the problem of optimization of the Boolean representation of a query in an on-line sequential database. This solution does not apply to the random access storage system dealt with in this paper. One of the provisions in Hanani's
paper is that a key cannot appear more than once in a query. This limits the scope of application of his solution.

The query is treated as a Boolean expression which is given a tree representation with AND given precedence over OR. Each attribute, value pair has associated with it two values, the time it takes, or rather the number of assembly language commands required, to check for the attribute, and the probability that a key is part of a record in the file. The tree is restructured according to an evaluation of these values and an optimal expression is found for which a search in the sequential file can be made.

Gedes and Hoffman (1979) comment on Hanani's article. The fact that no duplication of keys is allowed in a query is questioned because it is an unnatural situation. A solution that allows two key words in the same query to have the same attribute is proposed. This still limits the query, because the duplicated attribute must have different values for each occurrence. An algorithm that would lead to a further simplification of the query tree for this assumption was given.

Determining the minimum merge time for lists of records needed to satisfy a query based on equivalent Boolean expressions as represented in equivalent merge trees is approached by Lui (1976). In Lui's paper, it is assumed that the length of overlap or number of pointers in an overlap between two different lists is negligible, as compared with the length of the lists. Also, only two way merges are considered. Using Huffman merge trees, algorithms are developed to determine which Boolean expression is optimal for a query. Thus, the
order in which a merge should be made to minimize the total merge time is found.

Putkonen (1980) proposed that the query response time would be reduced by minimizing the time that is needed to merge address lists. In response to a query, in an inverted file system, normally the system first accesses the address lists that are associated with the attributes in a query. It then merges those address lists, then selects the records that satisfy the search logic. Her paper is an extension of Lui (1976), which contained algorithms for determining optimal merge trees for Boolean expressions. In Lui's paper the length of the intersection list is deemed negligible. In Putkonen’s paper, the length of this list is taken into account. To take advantage of the fact that the same address list may be accessed more than once, because the attribute associated with it occurs in more than one product, it is proposed that sometimes it will be advantageous for the union operation on the address list be performed before the intersection operation is performed. In this way it can be determined if time will be saved by taking into account an address list occurring in more than one product, which will decrease the merge time.

Trying to predict the amount of time a query will take is an approach that some researchers have used.

One of the factors in reducing query response time and the cost of accessing a query when records are grouped into blocks in secondary storage is knowing how many block accesses will be required to retrieve the records requested by the query. Several algorithms have been proposed to do this. Yao (1977) based his algorithm in part, on
the fact that a record can not be selected more than once when a retrieval is being made to satisfy a query. This was different from Cardenas (1973) who developed an algorithm that would allow a record to be selected more than once when a retrieval is being made to satisfy a query. Because Yao's algorithm was iterative and thus expensive to implement for a large number of records, Whang, Wiederhold, and Sagalowicz (1983) proposed a closed non-iterative formula that approximates Yao's exact iterative formula with acceptable accuracy.

Some queries specify records that have only values of some of their attributes requested in a query. These queries are called partial match queries. It is this type of query that is dealt with in this paper. For example, if a record has attributes, Social Security number, major, and grade point average, and the query specifies values for only the Social Security number and grade point average, it would be a partial match query. Another type of query is the closest match query. This type accesses records that may or may not have the exact same specifications as the query. In this case the user will usually specify a set number of records to be provided by the search. Those that comply most closely with the attributes given in the query will be retrieved.

Yu, Luk, and Sui (1978) devised a method of estimating the number of records that a partial match or a closest match query will satisfy. This would be useful if that number is very large and would enable the user to modify his query to avoid the prolonged search time. Being able to know the number of records satisfying a query, would be useful to statisticians or actuaries, who are more interested in the
number of records having a certain property rather than in the records themselves.
CHAPTER III

CURRENT QUERY OPTIMIZATION TECHNIQUES

Introduction to the Problem of Query Optimization

The problem of finding an optimal solution to the processing of a generalized database has proved to be a puzzle. There has been a missing factor involved that has caused all known strategies with a linear complexity to give non-optimal results.

A generalized database has in its index a list of keys along with a set of addresses of the first record in the linked list of records, all of which contain that key. These are called header nodes. Also in the index, associated with each key and its header nodes is the number of records in its linked lists.

When a query is made it can be transformed into the disjunctive form by using properties of Boolean algebra: the distributive, commutative, associative, and deMorgan's laws. This is done, not only because it is easier to conceive of queries in this form, but because it provides a grouping that can be systematically dealt with by algorithms that are used to find the optimal retrieval of records to satisfy the query.

The disjunctive form is a Boolean expression consisting of conjuncts that are ORed together. A conjunct consists of Boolean equalities that are ANDed together. An example of a conjunct is $K_1 \text{ AND } K_2 \text{ AND } \ldots \text{ AND } K_n$, where each $K_i$ is a key in the database. Thus the
disjunctive form would have the configuration:

\((K_1 \text{ AND } K_j \text{ AND } \ldots \text{ AND } K_1) \text{ OR } (K_m \text{ AND } K_n \text{ AND } \ldots \text{ AND } K_1) \text{ OR } \ldots \text{ OR } (K_q \text{ AND } K_r \text{ AND } \ldots \text{ AND } K_s)\). An example of a query that is in disjunctive form would be\(((\text{Major} = \text{Math}) \text{ AND } (\text{Minor} = \text{CS})) \text{ OR } ((\text{Major} = \text{CS}) \text{ AND } (\text{Minor} = \text{Engineering}))\).

Every record that satisfies a query should be retrieved. Since the query is in disjunctive form, if one conjunct is satisfied, then the records that satisfy that conjunct also satisfy the query. This is true because the conjuncts are ORed together. Every conjunct needs to be satisfied if all records that satisfy the query are going to be retrieved. It is also true that if a record satisfies a conjunct, then it must contain all of the keys in that conjunct. This is true because the keys in the conjunct are ANDed together.

One of the problems that arises when a database is in a generalized form with multilists, is that superfluous records are going to be retrieved when a query is being processed. This occurs because the records retrieved that contain one key of a conjunct may not contain all of the other keys in the conjunct.

For example, with the simple query \((K_1 \text{ AND } K_2)\), one could first retrieve all records that contain \(K_1\). This would be done by sequentially accessing all records in the linked list starting at the header node associated with the key, \(K_1\). Some of those records may not also contain the second key, \(K_2\). Only those with both keys will satisfy the query. If records containing \(K_2\) were accessed first and checked for the presence of \(K_1\), would time be saved? Looking at the length of the linked lists for the keys would help us choose. It
makes sense that the fewer records that are retrieved from secondary storage, the less time will be required for I/O. Thus when the query \((K_1 \text{ AND } K_2)\) is made, the first step is to check the length of the linked lists and to choose the key with the least length. There will still be records retrieved that are unneeded, but that number will be minimal. Selecting the key with the minimum linked list length for a conjunct is called the Prime Key Word method or PKW method.

The ORs in the query require a set of records from each conjunct to be retrieved. For example, if the query is \((K_1 \text{ OR } K_2)\) both linked lists for the keys will be retrieved from secondary storage. Records that are duplicated can be checked for and eliminated. This process is the same for more complex queries.

A more complex query such as \((K_1 \text{ AND } K_2) \text{ OR } (K_1 \text{ AND } K_3)\) could be satisfied in a number of ways. The key with the shortest linked list in each conjunct could be chosen and retrieved. If \(K_1\) has 40 records in its linked lists, \(K_2\) has 20 records, and \(K_3\) has 30, then from the first conjunct we could choose the key with the least records, which is \(K_2\). These records would be checked for the presence of \(K_1\). Using the same objective, from the second conjunct we would choose \(K_3\). The records associated with \(K_3\) would be retrieved and checked for the presence of \(K_1\). We would have then retrieved a total of 50 records from secondary storage, which is the sum of the number of records in \(K_2\) and \(K_3\). Three additional methods of satisfying queries are briefly discussed in the next section.
An Overview of Query Processing Methods

For the example above, \((K_1 \text{ AND } K_2) \text{ OR } (K_1 \text{ AND } K_3)\), note that \(K_1\) is common to both conjuncts. If it had been chosen, only 40 records would have had to be retrieved. This is one of the weaknesses of the FKW method. It does not always work when there is duplication of keys.

To take into consideration the fact that the same key can occur in more than one conjunct, we need a method that takes into consideration the frequency of occurrence of every key. Knowing this, we will know how many conjuncts the key occurs in because a key will not occur more than once in a conjunct. This second method is called the Minimum/Maximum technique or the Min/Max technique, for short. The strategy is to find a key with a minimum number of records associated with it that occurs a maximum number of times in the query. This can be found by deriving the ratio of number of records to the number of times it occurs in the query for each key and comparing them to find the minimum ratio. Even this strategy has shortcomings that will be demonstrated later in this chapter. Another factor must be taken into consideration to further optimize the retrieval process. This factor is the overlap of keys. This factor is taken into account in the query optimization method proposed in this paper and will be covered in the next chapter.

An additional method, called the Dual-to-MinDF algorithm, proposed by Kaminski (1984) will find the optimal set of keys to use for retrieval of the minimum number of records in the query. This algorithm guarantees that the fewest number of records will be retrieved.
from secondary storage, but has the drawback that its complexity is exponential and thus as the number of input variables in a query increases, the time that is needed to utilize the algorithm becomes very quickly prohibitive.

**COMPLEXITY**

One of the objectives in formulating an algorithm is to have it take as little time as possible to be executed. A predictive measure of this time is the complexity of the algorithm. While the number of times that the algorithm goes through a loop and the number and kind of calculations taking place in the loop are used as a measure of time, a count of the most expensive calculation occurring in the algorithm and how it changes with relation to the number of input variables is used to measure complexity.

An equation that expresses a worst case time of execution of the algorithm in terms of the input size, can be categorized as linear, polynomial, exponential, etc. This categorization is the complexity of the algorithm. The highest degree that the variable of input size takes is the degree of complexity of that algorithm. There may be coefficients and other terms in the equation of lesser degree that will effect the time of execution that should be taken into consideration when comparing algorithms of like complexity. Because a worst case is used to calculate complexity, some algorithms may prove to be more useful than could be foreseen from looking at the complexity.

The PKW technique has a linear complexity, while the Min/Max technique has polynomial complexity. These methods are limited in
their ability to provide an optimal solution to query optimization problems. The Dual-to-minDF method has a complexity that is exponential, but provides an optimal solution. The algorithm proposed in this paper has a complexity that is polynomial, while providing near optimal results.

Greedy and Dynamic Programming Methods

Both the PKW technique and the Min/Max techniques are examples of the greedy method. This method in general, requires there to be a process that decides whether a given input is feasible or not. If it is, that input is added to the solution set. The feasibility test in the PKW technique is a minimization of the number of records associated with the keys in a conjunct. The feasibility test in the Min/Max technique is minimization of the ratio of record number to frequency of all of the keys in a query.

The shortcoming of the greedy method is that the selection process is short sighted. The selection is based on local conditions and a sub-optimal solution is obtained unless all parameters of the problem are taken into consideration.

Another method which considers every possible combination and will find an optimal solution is the dynamic programming method used in the Dual-to-minDF method devised by Kaminski (1984). This method systematically checks every possible combination of keys and thus has an exponential complexity.

The method used to obtain an optimal solution in this paper is a greedy method that is used on several chosen subsets of the keys and
which ultimately results in a near optimal solution, because all relevant parameters are taken into consideration.

The Prime Key Word Technique

To further explain the first two query processing techniques and give a background that will help in understanding the query processing technique proposed in this paper, an expanded description of them, as well as examples that demonstrate their shortcomings are given in the remainder of this chapter.

The Prime Key Word, PKW, technique is the method often used in query optimization, even though it is the most inefficient method of the four discussed in this paper. This arises from the fact that the algorithm to detect the key with the least number of records associated with it is straightforward and simple. It is a loop in which 1) for each conjunct the PKW is found, then 2) each conjunct containing the chosen key is eliminated from further consideration. In simple queries with no redundancy of keys in conjuncts, this method, in fact, will give an optimal solution.

It is the redundancy of keys in conjuncts that causes this method to be non-ideal. For example, in the query shown in the last section, \((K_1 \text{ AND } K_2) \text{ OR } (K_1 \text{ AND } K_3)\), we saw when \(K_1\) has 40 records, \(K_2\) has 25 records, and \(K_3\) has 30 records the answer was non-ideal because, using the PKW technique, \(K_2\) and \(K_3\) were chosen for retrieval to satisfy the query. This caused 55 records to be retrieved. If \(K_1\) had been chosen, only 40 records would have been retrieved. Because the frequency was not taken into account, the solution is non-opti-
mal. This is one of the problems with using the greedy method to find an optimal solution. If all pertinent parameters are taken into consideration, the greedy method, which offers a linear complexity, can be optimal, if not, the solution is non-optimal.

Some additional examples which demonstrate the dependence of this method on the value of N for the keys are as follows. To simplify matters, the key subscript will designate the number of records containing that key.

Q1: \( (K_1 \text{ AND } K_4) \text{ OR } (K_2 \text{ AND } K_4) \) - \( (K_1, K_2) \) optimal
Q2: \( (K_3 \text{ AND } K_4) \text{ OR } (K_2 \text{ AND } K_4) \) - \( (K_2, K_3) \) non-optimal
Q3: \( (K_3 \text{ AND } K_4) \text{ OR } (K_5 \text{ AND } K_4) \) - \( (K_3, K_4) \) non-optimal
Q4: \( (K_5 \text{ AND } K_4) \text{ OR } (K_6 \text{ AND } K_4) \) - \( (K_4) \) optimal

Each of these queries could be solved by retrieving \( K_4 \) alone. Note that using the PKW method on Q1 will yield an optimal set of key values. In Q2, \( K_3 \) is substituted for the \( K_1 \) in Q1. Now the solution found with this method is non-optimal. This is because, in Q2, it is true that: \( \text{SUM}(N_2, N_3) > \text{SUM}(N_4) \). Note that in Q3, an extraneous key, \( K_3 \) is chosen. If the query had been stated \( (K_5 \text{ AND } K_4) \text{ OR } (K_3 \text{ AND } K_4) \), only \( K_4 \) would have been chosen from the first conjunct and in the second step of the loop, the second conjunct would have been eliminated, because \( K_4 \) covers it too. In Q4, the correct key is chosen, but only because it has minimum N value for both conjuncts.

It is clear that this algorithm is an unreliable one for choosing the best combination of keys for retrieval of records to satisfy a query. In this greedy method, there are missing parameters that
cause it to be less than optimal.

The Minimum/Maximum Technique

The Min/Max technique does take the frequency of occurrence of a key within a query into consideration. Again, this would be an optimal solution if keys occurred only once within a query. If this were so the PKW technique should be used for efficiency. On the other hand, if keys can be duplicated within a query, but not more than one duplicated key can appear within the same conjunct, this method, the Min/Max technique, is optimal.

This algorithm causes a count to be taken of the occurrences of a key within a single query. This frequency, $f_i$, is divided into the number of records associated with the key, $N_i$. The ratios, obtained for all of the keys in the query, are compared and the minimum ratio is found. The key associated with the ratio is put into the solution set and all conjuncts containing that key are eliminated, because they are covered by that key. The process is repeated for the keys in the remaining conjuncts.

An example will clarify the technique. Let us use the query:

$$(K_1 \text{ AND } K_2) \text{ OR } (K_1 \text{ AND } K_3)$$

again where $N_1 = 40$, $N_2 = 25$, and $N_3 = 30$.

The first step is to find the $N_i/f_i$ ratio for each key.

<table>
<thead>
<tr>
<th>key</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1$</td>
<td>$40/2 = 20$ <em>chosen</em></td>
</tr>
<tr>
<td>$K_2$</td>
<td>$25/1 = 25$</td>
</tr>
<tr>
<td>$K_3$</td>
<td>$30/1 = 30$</td>
</tr>
</tbody>
</table>
$K_1$ is chosen for the solution set for the first iteration. Because it covers both conjuncts, it is the solution set. In this case, the optimal solution was found.

Now, let us see if this method is successful for the examples of queries from the last section that did not have optimal results. These were:

- Q2: $(K_3 \text{ AND } K_4) \text{ OR } (K_2 \text{ AND } K_4)$
- Q3: $(K_3 \text{ AND } K_4) \text{ OR } (K_5 \text{ AND } K_4)$

In Q2, $K_4$ will be chosen first if we assume that when ratios are equal, the key with the highest frequency is chosen. Since both conjuncts are covered, the solution set is $(K_4)$, which is also optimal in this case.

It is when keys that are duplicated occur in the same conjunct, that this algorithm proves less than optimal. If the query is:

$$(K_3 \text{ AND } K_4) \text{ OR } (K_4 \text{ AND } K_5) \text{ OR } (K_5 \text{ AND } K_6)$$

and we again assume that the key number is the number of records with that key, the ratios in the first iteration will be:

- $K_3 : 3/1 = 3$
- $K_4 : 4/2 = 2 \ast \text{chosen}\ast$
- $K_5 : 5/2 = 2.5$
- $K_6 : 6/1 = 6$

$K_4$ is part of the solution set and because conjuncts 1 and 2 are covered by $K_4$, they are eliminated, leaving:

- $K_5 : 5/1 = 5 \ast \text{chosen}\ast$
- $K_6 : 6/1 = 6$

This yields $(K_4,K_5)$ as the solution set with 9 records retrieved, but
this is not an optimal solution because another set, \((K_3, K_5)\) is also a solution set and requires retrieval of 8 records.

Another problem encountered with the Min/Max method is illustrated by the following example:

\[(K_1 \text{ AND } K_4) \text{ OR } (K_4 \text{ AND } K_5)\]

In the first iteration the ratios are:

\[
\begin{align*}
\text{key} & \quad \text{ratio} \\
K_1 & : \quad 1/1 = 1 \quad *\text{chosen*} \\
K_4 & : \quad 4/2 = 2 \\
K_5 & : \quad 5/1 = 5 \\
\end{align*}
\]

and in the second iteration:

\[
\begin{align*}
\text{key} & \quad \text{ratio} \\
K_4 & : \quad 4/1 = 4 \quad *\text{chosen*} \\
K_5 & : \quad 5/1 = 5 \\
\end{align*}
\]

Thus the solution set is \((K_1, K_4)\). Yet the key, \(K_4\), covers both conjuncts and is the optimal solution set, \((K_4)\), causing only 4 records to be retrieved instead of 5. \(K_4\) is not chosen exclusively in the method shown because of the comparatively large difference in sizes of the number of records. There needs to be some way of checking to be sure an alternative is not better.

Another example of this problem is the query:

\[(K_1 \text{ AND } K_5) \text{ OR } (K_2 \text{ AND } K_5) \text{ OR } (K_3 \text{ AND } K_5)\]

The first iteration yields:

\[
\begin{align*}
\text{key} & \quad \text{ratio} \\
K_1 & : \quad 1/1 = 1 \quad *\text{chosen*} \\
K_2 & : \quad 2/1 = 2 \\
\end{align*}
\]
\[ K_3 : \frac{3}{1} = 3 \]
\[ K_5 : \frac{5}{3} = 1.67 \]
The conjuncts not covered by \( K_1 \) are:
\[(K_2 \text{ AND } K_5) \text{ OR } (K_3 \text{ AND } K_3)\]
which have the ratios:

\[
\begin{align*}
\text{key} & \quad \text{ratio} \\
K_2 : & \quad \frac{2}{1} = 2 \quad \text{*chosen*} \\
K_3 : & \quad \frac{3}{1} = 3 \\
K_5 : & \quad \frac{5}{2} = 2.5
\end{align*}
\]
This leaves \((K_3 \text{ AND } K_5)\) with ratios:

\[
\begin{align*}
\text{key} & \quad \text{ratio} \\
K_3 : & \quad \frac{3}{1} = 3 \quad \text{*chosen*} \\
K_5 : & \quad \frac{5}{1} = 5
\end{align*}
\]
Thus the solution set is \((K_1, K_2, K_3)\) with a total of 6 records retrieved. The set \((K_3)\) also covers all of the conjuncts and requires fewer retrievals. Again, the difference between the number of records for the keys was too great for the ratio to reflect an optimal combination.

We have seen the shortcomings of the current methods of optimizing the selection of keys for retrieval when satisfying a query in a generalized database. With this background, in the next chapter another method to optimize the key selection is given which is shown to be more effective.
CHAPTER IV

DESCRIPTION OF THE COMBINATION TECHNIQUE

The proposed technique for finding the optimal combination of keys, which is that combination of keys that will allow a query to be satisfied with the minimum number of records retrieved, will now be described.

The keys, contained in a query, are sorted in ascending order by the number of records in their linked lists, and a chart is created with the keys arranged in ascending order and indicating in which conjunct(s) the key is present. Then systematically checking the keys and their combinations for a cover, a combination of keys that has the potential to be the minimum number of records is produced. Also, if a list of keys with frequency greater than one is created and each of these keys, which are potential members of the final solution set, are systematically combined with the other keys of the query, other combinations of keys will be produced that have the potential of being the optimal set of keys. Comparing the number of records associated with each of these combinations of keys and choosing the combination with the minimum number of records associated with it will provide us with the optimal set of keys.

It is the way in which the keys are checked that makes this method close to optimal. The first part of this method is described as follows:

The first key in the sorted list, which is the one with the
smallest number of records associated with it, is checked to see if it covers all of the conjuncts in the query, which means that it occurs in each of the conjuncts. If it does, we are done with the first iteration.

If it does not, it is added to the preliminary solution set and the second key is checked to see if it covers conjuncts that the first key does not cover. If so, it is added to the preliminary solution set. If not, the next key is checked. When a key is added to the preliminary solution set, the combination is examined to see if it forms a cover for all of the conjuncts.

The last key added to the preliminary solution set must contain a key that covers a conjunct not covered by any other key, otherwise it would not have been added to the set. This key is placed into the proposed final solution set and a check is made to see if it is a cover for all of the conjuncts. If not, this final key is made the first key in a subsequent preliminary solution set. The first key in the sorted list is again checked to see if it covers a conjunct not covered by the key(s) in the preliminary solution set. If so, it is added to this preliminary solution set. This is repeated until all conjuncts are covered by the keys in this preliminary solution set. These steps are repeated until all conjuncts are covered by the keys in the proposed final solution set.

If we define:

- \( k \) - number of keys in the query
- \( m \) - number of conjuncts in the query
- \( K_1, K_2, \ldots, K_n \) - keys in the query
PS - preliminary solution set
PPS - proposed final solution set,
we can state the algorithm so far in the following form:
Sort keys according to number of records associated with key
Arrange keys in increasing order
PPS set to zero
While not done
Begin
PS set equal to PPS
While not complete cover AND i = k do
Begin
If \( K_i \) covers a conjunct not covered by PS then
add \( K_i \) to PS
If PS covers all conjuncts then
complete cover = true
else
i = i + 1
End
Add \( K_i \) to PPS
if PPS is a complete cover then
done = true
End
This portion of the technique does not produce an optimal solution in every case, but upon testing it on 100 random queries along with the Prime Key Word technique and the Min/Max technique, the accuracy of results for this portion of the Combination technique...
proved to be better than that of the Prime Key Word technique and equal to that of the Min/Max technique. More will be said in Chapter Five about the testing done on these random queries, along with the complexity of these algorithms.

This portion of the Combination technique takes into account the duplication of keys as well as their overlap in conjuncts, but because it does not take the frequency of the keys in the query into account, it is not optimal. Thus in a query: \((K_2 \text{ AND } K_4) \text{ OR } (K_3 \text{ AND } K_4) \text{ OR } (K_5)\) the chart created after the keys are sorted would be:

<table>
<thead>
<tr>
<th>conjuncts</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>(K_2)</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(K_3)</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(K_4)</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

The first preliminary solution set would be: \((K_2, K_3, K_5)\) with final solution set \((K_5)\). The next preliminary solution set would be \((K_5, K_2, K_3)\) with proposed final solution set \((K_5, K_3)\). The last preliminary solution set would be \((K_5, K_3, K_2)\) and the proposed final solution set would be \((K_5, K_3, K_2)\). This would not be optimal because 10 records would be retrieved. If \((K_5, K_4)\) had been chosen, only 9 records would have to be retrieved. The greater frequency of \(K_4\) was ignored.

The way that the frequency of the keys is considered in the Combination technique is to list all keys with frequency over one and use each of these keys in turn as the initial key in the first pre-
liminary solution set and in the final solution set. The algorithm stated above is used to find a proposed final solution set for each of these keys. For each proposed final solution set, the sum of records is found and compared to the minimum sum found up until then. The proposed final solution set with the least records is saved and a final solution set is found.

Using the example given above, \((K_2 \text{ AND } K_4) \text{ OR } (K_3 \text{ AND } K_4) \text{ OR } K_5\), if we listed the keys with frequency greater than one, we would have \((K_4)\). Using \(K_4\) as the initial key in the preliminary and proposed final solution sets, we would have \((K_4, K_5)\) as both the proposed final and preliminary solution sets after the first iteration. This proposed final solution set is a complete cover and is an alternative solution to \((K_5, K_3, K_2)\). Comparing them, \((K_5, K_4)\) would be chosen as minimum and the optimal solution has been found.

A more extensive example follows, showing the preliminary solution set, PS; proposed final solution set, PFS; and final solution set, FS; at each step. The query is:
\[(K_5 \text{ AND } K_8) \text{ OR } (K_3 \text{ AND } K_4 \text{ AND } K_6) \text{ OR } (K_5 \text{ AND } K_6) \text{ OR } (K_2 \text{ AND } K_4 \text{ AND } K_8)\]. The chart would be:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
K_2 & x \\
K_3 & x \\
K_4 & x & x \\
K_5 & x & x \\
K_6 & x & x \\
K_8 & x & x \\
\end{array}
\]
The set of keys with frequency greater than one is \((K_4, K_5, K_6, K_8)\).

First we find the proposed final solution set using the algorithm with PS and PFS empty. Each line represents an iteration.

- **PS**\((K_2, K_3, K_5)\)  **PFS**\((K_5)\)
- **PS**\((K_5, K_2, K_3)\)  **PFS**\((K_5, K_3)\)
- **PS**\((K_5, K_3, K_2)\)  **PFS**\((K_5, K_2, K_2)\)  **FS**\((K_5, K_3, K_2)\)  number of records 10

Using each key with frequency greater than two as the initial member of **PS**, we obtain:

- **K_4**  **PS**\((K_4, K_5)\)  **PFS**\((K_5)\)
  - **PS**\((K_5, K_4)\)  **PFS**\((K_5, K_4)\)  **FS**\((K_5, K_4)\)  number of records 9
- **K_5**  **PS**\((K_5, K_2, K_3)\)  **PFS**\((K_5, K_3)\)
  - **PS**\((K_5, K_3, K_2)\)  **PFS**\((K_5, K_3, K_2)\)  **FS**\((K_5, K_4)\) remains minimum
- **K_6**  **PS**\((K_6, K_2, K_5)\)  **PFS**\((K_6, K_5)\)
  - **PS**\((K_6, K_5, K_2)\)  **PFS**\((K_6, K_5, K_2)\)  **FS**\((K_5, K_4)\)
- **K_8**  **PS**\((K_8, K_3, K_5)\)  **PFS**\((K_8, K_5)\)
  - **PS**\((K_8, K_3, K_3)\)  **PFS**\((K_8, K_3, K_2)\)  **FS**\((K_5, K_4)\)

The solution is \((K_4, K_5)\).

The potential worst increase in time for using the algorithm on keys with frequency greater than one is a factor of the number of keys in the query, but in many cases the extra time required will be eclipsed by the time saved by not retrieving hundreds of extra records.

Though this method is very accurate, it is not optimal, which is illustrated by the following example.

\((K_2 \text{ AND } K_5) \text{ OR } (K_4 \text{ AND } K_7) \text{ OR } (K_5 \text{ AND } K_7) \text{ OR } (K_4 \text{ AND } K_5) \text{ OR } (K_6 \text{ AND } K_7)\)
The first part of the technique would yield \((K_4, K_5, K_6)\) with a total of 15 records retrieved. The complete technique would produce \((K_7, K_2, K_4)\) with 13 records retrieved, but the optimal solution is \((K_7, K_5)\). Instead of placing all keys with frequency greater than one in the same set, separate sets for each different frequency greater than one in the query could be created. Then considering in turn each key in a set combined with only members of that and those keys with frequency greater than that set, we would eliminate this problem. To do so would take more time than is worth while, because the Combination technique as it now stands has a very high reliability.
CHAPTER V

TESTING OF THE RETRIEVAL MINIMIZATION TECHNIQUES

The Algorithms and Their Reliability

In an attempt to measure reliability of the three techniques for reducing record retrieval to satisfy a query in a generalized data base, three algorithms were implemented using queries that were generated using the PASCAL RANDOM function as input. One hundred of these queries, that had keys with from one to twenty records associated with them, were generated for the test. This program and the queries generated can be found in Appendix A. These keys were incorporated in queries in the disjunctive form which allowed up to seven conjuncts of two to five keys in each conjunct. The reliability of a method is defined here as the percentage of instances that an algorithm produces an optimal solution.

The algorithms used were as follows and the actual programs can be found in Appendix B.

Let: $k$ - number of keys in the query
$m$ - number of conjuncts in the query
$s$ - number of keys in the largest conjunct in the query
$(K_1, K_2, \ldots, K_k)$ - keys in the query
$f_i$ - frequency of $k_i$
$r_i$ - number of records associated with $k_i$

For example, in the query $(K_1 \text{ AND } K_2) \text{ OR } (K_2 \text{ AND } K_3 \text{ AND } K_4)$

$k = 4$,
m = 2,
s = 3,
(K_1, K_2, K_3, K_4) is the set of keys in the query,
f_1 = 1, f_2 = 2, f_3 = 1, f_4 = 1,
r_1 = number of records associated with K_1,
r_2 = number of records associated with K_2,
r_3 = number of records associated with K_3,
r_4 = number of records associated with K_4.

Prime Key Word algorithm:

For i = 1 to m do

Begin

If conjunct is not eliminated then

For j = 1 to s do

Find minimum key in conjunct;
Place minimum key in solution set;
Eliminate covered conjuncts;

End;

Minimum/Maximum algorithm:

While all conjuncts are not covered

Begin

(* Find frequency of keys in query *)

For i = 1 to k do

Begin

For j = 1 to m do

Begin

If conjunct not covered do

End

End

End
For $h = 1$ to $s$ do
Begin
If $Key(i) = position(j,h)$ then
\[ f(i) = f(i) + 1; \]
End;
End;
End;
Find $r(i)/f(i)$ for each $K(i)$;
Find minimum $r(i)/f(i)$;
(* Eliminate conjuncts with chosen key *)
For $j = 1$ to $m$ do
Begin
For $h = 1$ to $s$ do
If $key(j,h) = \text{chosen key}$ then
Eliminate conjunct $j$;
End;
End.
End.

Combination Technique algorithm:
Sort $k$ keys;
Find frequency of the $k$ keys;
Create $FG_1$; (* set of keys with frequency greater than one *)
Min = 99999;
(* Combination algorithm *)
For $i = 0$ to $f$-num do (* for all keys in $FG_1$ *)
Begin
\[ PFS(l) = FG_1(i); \]
While NOT Done

Begin

PS set to PFS;

While NOT complete cover AND j = k do

Begin

If k(j) covers a conjunct not covered by PS then

Add k(j) to PS;

If PS covers all conjuncts then

complete cover = true;

Else

j = j + 1;

End;

Add k(j) to PFS;

If PFS covers all conjuncts then

Done = true;

End;

If SUM(PFS) < Min then PS = PFS;

End.

When 100 of these random queries were used with these algorithms, it was found that the following percentage of optimal results was obtained.

<table>
<thead>
<tr>
<th>reliability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime Key Word</td>
<td>66%</td>
</tr>
<tr>
<td>Minimum/Maximum</td>
<td>79%</td>
</tr>
<tr>
<td>Combination</td>
<td>99%</td>
</tr>
</tbody>
</table>

Appendix C contains the actual results for each of the queries.
using the three algorithms. The actual programs used are contained in Appendix B.

Complexity of Retrieval Minimization Techniques

In order to compute complexity, we must find an expression expressed in terms of the number, \( n \), of input symbols, in this case keys, that is determined from the worst-possible case for the algorithm. This \( n \) is not the number of keys, \( k \), but a count of symbols that are input. The query \((K_1 \text{ AND } K_2) \text{ OR } (K_2 \text{ AND } K_3)\) has 4 input symbols that are used by the three algorithms. Thus \( n = 4 \). Because the parentheses and Boolean symbols merely indicate position within an array in the programs, they are not counted. The variable, \( k \), has the value 3 in this example.

We want to express \( n \) in terms of \( k \). We know from combinatorial theory that "the total number of combinations of \( n \) distinct things, any number at a time is \( 2^n \)" (Riordan 1958). The worst possible case would include every possible combination of the \( k \) keys. To find how many times a certain key could occur in that worst case query, we observe that if a chosen key occurs with every combination of the remaining \( k-1 \) keys, there would be \( 2^{n-1} \) combinations for it to occur with. Because there are \( k \) keys that this is true for, our formula for the worst possible case query is \( n = k^2 k^{-1} \).

For example, with \( k = 3 \) we can have every combination of the 3 keys:

\[
(K_1) \text{ OR } (K_2) \text{ OR } (K_3) \text{ OR } (K_1 \text{ AND } K_2) \text{ OR } (K_1 \text{ AND } K_3) \text{ OR } (K_2 \text{ AND } K_3) \quad \text{OR}
\]
$(K_1 \text{ AND } K_2 \text{ AND } K_3)$

$K_1$ occurs 4 times, as does $K_2$ and $K_3$. This is $2^{3-1} = 4$. There are a maximum of 12 occurrences of keys to be input in this query. This is $3 \times 2^{3-1} = 12$ occurrences.

An expression for $k$ in terms of $n$ can be derived from $n = k^2 k^{-1}$.

$$n = k^2 k^{-1}$$
$$n > 2^{k-1}$$

$$\log_2 n > \log_2 2^{k-1}$$

$$\log_2 n > k - 1$$

$$\log_2 n + 1 > k$$

We can substitute $\log_2 n + 1$ for $k$ to arrive at the order of complexity of our algorithms, because we are not decreasing the value of the order of complexity by doing so.

Looking at the PKW algorithm we find that the complexity will be related to $m \times s$, because there are two FOR loops, one with $m$ repetitions and the other with $s$ repetitions. $m \times s$ is the number of conjuncts times the number of keys that are in the conjunct with the most keys in it. The worst case for $m$ would be the largest number of combinations possible given a $k$. We have seen that this is $2^k$. The largest value that $s$ could have would be $k$, every key. Substituting these values gives us:

$$m \times s = k^2 k$$

$$m \times s = 2n \text{ because } n = k^2 k^{-1}$$

$$2n = k^2 k$$

Thus the order of complexity for the Prime Key Word technique is $O(n)$.  

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The order of complexity for the Minimum/Maximum algorithm can be seen to be dependent on the finding of the frequency of the keys. This portion of the algorithm is done at most \( m \) times, assuming that at each iteration only one conjunct is covered. Within the frequency portion, there are three FOR statements, the first is done \( k \) times, the second contains an if statement that checks for coverage of the conjunct, so it is done a maximum of \( m-1 \) times, and the third is done \((m-1) \sum_{i=0}^{m-1} \) times. This results in the statement \( \text{mks}(m-i) \) that can be used to determine the complexity of this algorithm. Expressing this in terms of \( n \) as in the PKW algorithm would yield:

\[
\begin{align*}
\text{mks}(\sum_{i=0}^{m-1} m-i) & = 2^k k (\sum_{i=0}^{2^k-1} i) \\
& = 2^k k 2^{2^k} - \sum_{i=0}^{2^k-1} i \\
& = 2^k k 2^{2^k} - 2^k k^2 (2^k + 1) - 2^k \\
& = 2^k k 2^{2^k} - 2^k k^2 (2^k + 1 - 2^k) \\
& = 2^k k 2^{2^k} - 2^k k^2 (2^{2k-1} + 2^{k-1} - 2^k) \\
& = 2^k k 2^{2^k} - 2^k k^2 (2^{2k-1} + 2^{k-1} + 2^{2k-2}) \\
& = (2n)^{2^k} - (2n)^{2^k-1} - (2n)^2 + (2n)^2 \\
& = 4n^{2^k \log_2 n+1} - 4n^{2^k \log_2 n} - 2n^2 + 4n^2 \\
& = 8n^2 n - 4n^2 n + 2n^2 \\
& = 4n^3 + 2n^2
\end{align*}
\]

This gives a complexity of \( O(n^3) \).

The Combination technique algorithm would have its complexity
based on the combination step where the loop that is finding keys to add to the preliminary solution set, tests keys only down to the last key added. Thus this loop would proceed through \( k-1 \) keys in the worst case with each of the \( m \) conjuncts checked for coverage for each of the keys. A worst case would be \( k \) iterations of this loop in the algorithm, because it is possible for all of the keys to have frequency greater than one. Thus we would have:

\[
\begin{align*}
&\sum_{i=0}^{k-1} km^i \\
= &\sum_{i=0}^{k-1} k^2 m^i \\
= &2n(k^2 - \sum_{i=0}^{k-1} i) \\
= &2n(k^2 - \frac{k(k+1) - k}{2}) \\
= &2nk^2 - nk^2 - nk + 2nk \\
= &nk^2 + nk \\
= &n(\log_2{n} + 1)^2 + n(\log_2{n} + 1) \\
= &n(\log_2{n})^2 + 2n\log_2{n} + n + n\log_2{n} + n \\
= &n(\log_2{n})^2 + 3n\log_2{n} + 2n \\
\end{align*}
\]

Thus the order of complexity of the Combination Technique algorithm is \( O(n(\log_2{n})^2) \).
CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

Query processing is becoming an increasingly important aspect of database systems, as these systems increase in size and are used more heavily by businesses, schools, and any individual or group desiring access to a body of data.

Attention has focused on many aspects of this topic. Some of these approaches used to reduce query processing time are analyzing the query and reducing it by various methods to an equivalent Boolean expression which will cause fewer records to be retrieved from the database, designing the database by analyzing the given set of data, creating hashing methods of addressing within the database to reduce query response time. When using a database that conforms to a generalized database and is not an inverted file structure, as described by Hsiao and Harary (1970), another matter that has great impact on query processing time is the method used to eliminate the retrieval of records that do not satisfy the query.

The methods used commonly, the Prime Key Word technique, has a rather low reliability and an order of complexity of $O(n)$. This method utilizes a greedy method to obtain a feasible solution, as does the Minimum/Maximum technique with a higher reliability and complexity of $O(n^3)$. The method proposed by this paper, the Combination technique, is also a greedy method, but it takes into consideration an aspect of queries that neither the Prime Key Word nor the Minimum
Maximum technique take into account. This is the overlap of keys, the fact that keys with frequency greater than one can occur in the same conjunct. While the Combination technique proposed in this paper is not optimal, it does have very high reliability and a complexity of $O(n(\log_2 n)^2)$. This gives the technique a general usefulness that is not obtained by one that utilizes a dynamic programming method with its exponential order of complexity.

RECOMMENDATIONS FOR FURTHER RESEARCH

Because of the relatively low complexity and high reliability of the Combination technique described in this paper, the expansion of the method suggested in Chapter Four, was not developed. It would be interesting to see if this method is optimal, how much more time would be needed to use this expanded version, and what its complexity would be.

The Prime Key Word technique requires a relatively short time for implementation when compared to the Combination technique, while its reliability is low when the frequency of any key in the query is greater than one. If a test for frequency of each key was made upon the input of a query, the choice of using the Prime Key Word technique for those queries with no key having frequency greater than one could be made and the Combination technique could be used for the others. This would reduce the time for processing the query considerably, as compared to using only the Combination technique.

Another area of concern in query processing regarding the retrieval of records from secondary storage is the topic of optimal
block retrieval. The grouping of records in blocks would have an im-
pact on the number of blocks accessed to retrieve all of the records
required by the query. The probability of records being in the same
block would need to be calculated considering the number of records
associated with a key.

It is normal for some keys within a data base to have a higher
probability of being requested than others. In a database that is
structured to allow easier access to those keys with higher probabil-
ity of use, these probabilities would need to be taken into consider-
atation when choosing from proposed final solution obtained from the
Combination technique and the final set of keys would be chosen on
this basis.

There are many aspects to consider when optimizing query proc-
essing techniques. Among these is a consideration of choice of keys
for retrieval to satisfy that query. This study shows a method with
high reliability and low complexity and so contributes to the body of
research on this topic.
PROGRAM MAKE(INPUT/OUTPUT);
VAR I, J, N, K, L, M: INTEGER;
  CON_NUM, KEY_NUM, S, A: INTEGER;
  KEY: ARRAY[1..10] OF INTEGER;
  NO_DUP: BOOLEAN;
PROCEDURE GET RANDOM(VAR VALUE : INTEGER);
VAR R, S: REAL;
BEGIN
  S := RANDOM(R) * 10.0;
  VALUE := TRUNC(S);
END;
BEGIN
  FOR I := 1 TO 100 DO
    BEGIN
      GET RANDOM(CON_NUM);
      CON_NUM := CON_NUM MOD 7 + 1;
      FOR J := 1 TO CON_NUM DO
        BEGIN
          GET RANDOM(KEY_NUM);
          KEY_NUM := (KEY_NUM MOD 4) + 2;
          WRITE(',');
          A := 1;
          N := 1;
          WHILE (N <= KEY_NUM) DO
            BEGIN
              NO_DUP := TRUE;
              GET RANDOM(M);
              M := M + 1;
              GET RANDOM(L);
              K := M + L;
              FOR S := 1 TO A DO
                IF (KEYCS3 = K) THEN
                  NO_DUP := FALSE;
              IF NO_DUP THEN
                BEGIN
                  N := N + 1;
                  A := A + 1;
                  KEY[ ] := K;
                  IF (K >= 10) THEN
                    WRITE(K:2)
                  ELSE
                    WRITE(K:1);
                  IF (N <= KEY_NUM) THEN
                    WRITE('"');
                  END;
              END;
              WRITE('"');
              IF (J < CON_NUM) THEN
                WRITE('v')
              ELSE
                WRITE('v');
            END;
          END;
        END
    END
END.
Appendex B

Programs for Testing the Three Algorithms
PROGRAM RETRIEVE INPUT, OUTPUT; * 

CONST
SKIP = 35;
MAXSIZEOFQUERY = 35;
MAXNOFKEYS = 21;
MAXDISJ = 7;
MAXFUN = 256;
MAXSIZE = 2006;
MAXOFCOMBINEDNOS = 20;
FINAL_DISJ_IMPlicants = 10; 

TYPE
QUERY_TYPE = ARRAY[1..MAXSIZEOFQUERY] OF INTEGER;
STORAGE = ARRAY[1..MAXDISJ,1..MAXNOFKEYS] OF INTEGER;
SELCK_TYPE = ARRAY[1..MAXNOFKEYS] OF INTEGER;
TLOCATIONS = ARRAY[1..MAXNOFKEYS] OF REAL;
COVERTYPE = ARRAY[1..MAXNOFKEYS,1..MAXDISJ] OF BOOLEAN;
FCOVERTYPE = ARRAY[1..MAXNOFKEYS] OF BOOLEAN;

VAR
QUERY : QUERY_TYPE;
COVER : COVERTYPE;
KEYS_NO : INTEGER;
WORKING_STORAGE : STORAGE;
FREQUENCY,
KEYS_NO,
TEMP,
RECORDS,
SELECTED_KEY : SELCK_TYPE;
K,
P,
S,
Y,
L,
KEY,
SUM,
COLUMN,
NO_OF_VARIABLES,
Q_SIZE,
RK_SIZE,
SK : INTEGER;
CH : CHAR;
ARC : INTEGER;
KEYS : INTEGER;
Q_COVER, COVER, FOUND, MIN_COVER, TEST_COVER : BOOLEAN;
FINAL_KEYS, SELCK_TYPE;
KEY_COPY, SELCK_TYPE;
SELCK_COVER, FCOVERTYPE;
FREQUENCY,
IFREQ, FREQN : SELCK_TYPE;
P,Q : INTEGER;
MINN, INTEGER;
FIRST / SECOND : INTEGER;
COUNT : INTEGER;

(* PROCEDURE INITIALIZE *)
(* --------------------- *)

* Parts of this program were adapted from one by Jamel Ali Mathanna
PROCEDURE INITIALIZE;
VAR I : INTEGER;
BEGIN
FOR I := 1 TO MAXSIZEOFQUERY DO
  QUERY[I] := 0;
FOR I := 1 TO MAXNOFOFKEYS DO
  KEY_NOLI[I] := 0;
END;

(* *********************** *)
(* PROCEDURE INIT_WORKING *)
(* *********************** *)
PROCEDURE INIT_WORKING;
VAR K : INTEGER;
BEGIN
FOR I := 1 TO MAXDISJ DO
  FOR K := 1 TO MAXNOFOFKEYS DO
    WORKING_STORAGE[I,K] := 0;
END;

(* *********************** *)
(* PROCEDURE PRINT_HEADING *)
(* *********************** *)
PROCEDURE PRINT_HEADING;
BEGIN
  WRITELN(SKIP:4/*", PRIME KEY WORD TECHNIQUE, * MINIMUM/MAXIMUM TECHNIQUE * COMBINATION TECHNIQUE",*/);  
  WRITELN(SKIP:4/*", # OF --- # OF --- # OF --- # OF --- */);  
  WRITELN(SKIP:4/*", KEYS FOR RETRIEVAL | RECORDS * KEYS FOR RETRIEVAL | RECORDS * KEYS FOR RETRIEVAL | RECORDS */);  
END;

(* *********************** *)
(* PROCEDURE TOTAL RECORDS *)
(* *********************** *)
PROCEDURE TOTAL_RECORDS(VAR TEMP : SELCK_TYPE ; VAR Y,SUM : INTEGER);
VAR I : INTEGER;
BEGIN
  SUM := 0;
  FOR I := 1 TO Y DO
    BEGIN
      SUM := SUM + RECORDS[TEMP[I]];
    END;
END;
```haskell
(* PROCEDURE READ RECORDS *)

PROCEDURE READ_RECORDS;

VAR
  I : INTEGER;
BEGIN
  WHILE NOT EOF DO
    READ(N,I);
    RECORDS[I] := N;
  END;
END;

(* PROCEDURE READ_QUERY *)

PROCEDURE READ_QUERY;

VAR
  DONE : BOOLEAN;
BEGIN
  DONE := FALSE;
  READLN;
  WHILE (NOT EOF) AND (NOT DONE) DO
    BEGIN
      SIZE := SIZE + 1;
      READ(KEY1);
      READ(K1);
      QUERY[SIZE] := KEY1;
      TEMPLATE[KEY1] := KEY1;
      IF TEMPLATE[KEY1] = 1 THEN BEGIN
        SIZE := SIZE + 1;
        READ(K2);
        IF K2 = 'A' THEN BEGIN
          DONE := TRUE;
        END;
      END;
    END;
    READLN;
  END;
END;

(* PROCEDURE SET_RECORDS *)

PROCEDURE SET_RECORDS;
VAR
  I : INTEGER;
BEGIN
  FOR I := 1 TO MAXNOFKEYS DO
    BEGIN
      IF (TEMPLATE[I] = 1) THEN BEGIN
        RECORDS[I] := TEMPLATE[I];
      END;
    END;
END;

(* PROCEDURE READ_INFC *)

PROCEDURE READ_INFC;
BEGIN
  READ_QUERY;
  SET_RECORDS;
END;

(* PROCEDURE SET_KEYS *)

PROCEDURE SET_KEYS;
VAR
  I : INTEGER;
BEGIN
  L := 1;
  FOR I := 1 TO MAXNOFKEYS DO
    BEGIN
      IF (TEMPLATE[I] = 1) THEN BEGIN
        KEY_NOCL := TEMPLATE[I];
        FREQNCL := TEMPLATE[KEY_NOCL];
      END;
    END;
END;

(* PROCEDURE MAKE_FRESET *)

PROCEDURE MAKE_FRESET;
VAR
  I : INTEGER;
BEGIN
  E := 0;
  FOR I := 1 TO L DO
    BEGIN
      IF (FREQNCL > 1) THEN BEGIN
        F := F + 1;
      END;
    END;
END;
```
BEGIN
  WRITE("v");
  COUNT := COUNT + 2;
END;
ELSE BEGIN
  IF (I <> 0) THEN
    BEGIN
      WRITE("v");
      COUNT := COUNT + 2;
    END;
END;
WRITE('(') ;
COUNT := COUNT + 1;
END;
ELSE
BEGIN
WRITE(')') ;
COUNT := COUNT + 1 ;
END;
END;

(* PROEDURE COPY QUERY *)
(* --------------------- *)

PROCEDURE COPY_QUERY_TO_WORKING_STORAGE(VAR S/WS_SIZE : INTEGER) ;
VAR
MX , I : INTEGER ;
BEGIN
MX := 0 ;
S := 0 ;
WS_SIZE := 1 ;
FOR I := 1 TO (Z_SIZE-1) DO
BEGIN
IF (QUERY[I] = 0) THEN
BEGIN
S := 0 ;
WS_SIZE := WS_SIZE + 1
END ;
ELSE
BEGIN
S := S + 1 ;
WORKING_STORAGE[WS_SIZE/S] := QUERY[I] ;
IF (S >= MX) THEN
BEGIN
MX := S ;
END ;
END ;
END ;
S := MX ;
END ;

(* PROEDURE MIN RECORD KEY *)
(* ------------------------ *)

PROCEDURE FIND_KEY_WITH_MIN_RECORDS(VAR I/K/SK,S/MIN : INTEGER) ;
VAR
J : INTEGER ;
BEGIN
WHILE (WORKING_STORAGE[K/I] <> 0) AND (I <= S) DO
BEGIN
IF (RECORDS[WORKING_STORAGE[K/I]] <= MIN) THEN
BEGIN
MIN := RECORDS[WORKING_STORAGE[K/I]] ;
SK := WORKING_STORAGE[K/I] ;
END ;
I := I + 1 ;
END ;
J := J + 1 ;
END ;
J := k ;
END ;
I := I + 1 ;
END ;
IF (J > 0) THEN
WORKING_STORAGE[J-1] := 0 ;
END ;

(* * * * * * * * * * * * * * * * * *)
(* PROCEDURE FIND COVERS *)
(* * * * * * * * * * * * * * * * * *)
PROCEDURE FIND_COVERS(VAR sk, s, ws_size : INTEGER) ;
VAR
I : INTEGER ;
DONE : BOOLEAN ;
BEGIN
FOR I := 1 TO ws_size DO
BEGIN
IF (WORKING_STORAGE[I,1] <> 0) THEN
BEGIN
DONE := FALSE ;
L := 1 ;
WHILE (NOT DONE) AND (WORKING_STORAGE[I,L] <> 0) AND (L <= s) DO
BEGIN
IF (WORKING_STORAGE[I,L] = sk) THEN
BEGIN
DONE := TRUE ;
WORKING_STORAGE[I,1] := 0
END
ELSE
L := L + 1 ;
END
END ;
END ;
BEGIN
I := 1 ;
TAB := INTEGER ;
BEGIN
TAB := FIRST - COUNT ;
WRITE(Skip:TAB,"*"');
COUNT := FIRST + 2 ;
FOR I := 1 TO Y DO
BEGIN
WRITE("K") ;
IF (SELECTED_KEY[I] >= 10) THEN
BEGIN
WRITE(SELECTED_KEY[I]:2,"");
COUNT := COUNT + 4;
END;
ELSE
BEGIN
WRITE(SELECTED_KEY[I]:1,"");
COUNT := COUNT + 3;
END;
TAB := SECOND - COUNT;
WRITE(\$12:TAB,"1");
WRITE(\$13:SUM,"3");
COUNT := SECOND + 5;
END;

(* PROCEDURE SET FREQUENCY*)

PROCEDURE SET_FREQ;
VAR
  I : INTEGER;
BEGIN
  FOR I := 1 TO MAXNOFKEYS DO
    FREQUENCY[I] := 0;
END;

(* PROCEDURE FREQUENCY *)

PROCEDURE FREQUENCY(VAR S,L,WS_SIZE : INTEGER);
VAR
  M,
  I,
  K : INTEGER;
BEGIN
  FOR I := 1 TO L DO
  BEGIN
    R := 0;
    FOR K := 1 TO WS_SIZE DO
      BEGIN
        IF (WORKING_STORAGE[K,I] <> C) THEN
          FOR M := 1 TO S DO
            BEGIN
              IF (KEY_NO[I] = WORKING_STORAGE[K,M]) THEN
                R := R + 1;
            END;
        END;
      END;
    FREQUENCY[I] := R;
  END;
END;

REPRODUCED WITH PERMISSION OF THE COPYRIGHT OWNER. FURTHER REPRODUCTION PROHIBITED WITHOUT PERMISSION.
PROCEDURE N_DIV_(VAR L : INTEGER ; VAR TEMP2 : T_LOCATIONS) ;

VAR
I : INTEGER ;
A : REAL ;
BEGIN
FOR I : = 1 TO L DO
IF (FREQUENCY[I] <> 0) THEN
BEGIN
A := RECORDS[KEY_NO[I]] ;
B := FREQUENCY[I] ;
TEMP2[I] := A / B ;
ELSE
TEMP2[I] := 999999.0 ;
END ;
END ;

PROCEDURE FIND_MIN_MAX(VAR SK, L : INTEGER ; VAR MIN : REAL ; VAR TEMP2 : T_LOCATIONS) ;

VAR
I : INTEGER ;
BEGIN
FOR I : = 1 TO L DO
BEGIN
IF (TEMP2[I] <= MIN) THEN
BEGIN
MIN := TEMP2[I] ;
SK := KEY_NO[I] ;
END ;
END ;

PROCEDURE PRIME_KEY_TECHNICUE(VAR Y, S, WS_SIZE : INTEGER) ;

VAR
K, X, I, MIN : INTEGER ;
BEGIN
COPY QUERY TO WORKING_STORAGE(S, WS_SIZE);
PRINT_INPUT;
Y := 0;
FOR X := 1 TO WS_SIZE DO
BEGIN
  IF (WORKING_STORAGE[X, 1] <> 0) THEN
  BEGIN
    K := X;
    MIN := 0.999;
    I := 1;
    FIND_KEY_WITH_MIN_RECORDS(I, K, SK, S, MIN);
    Y := Y + 1;
    SELECTED_KEY[Y] := SK;
    FIND_COVERS(SK, S, WS_SIZE);
  END;
END;
TOTAL_RECORDS(SELECTED_KEY, Y, SUM);
PRINT_OUT(SELECTED_KEY, Y, SUM, 47, 52);
END;

(* PROCEDURE MIN/MAX TECH. *)
(* ------------------------ *)
PROCEDURE MIN_MAX_TECHNIQUE(VAR Y, S, WS_SIZE : INTEGER);
VAR
  I : INTEGER;
  TEMP2 : T_LOCATIONS;
  MIN : REAL;
BEGIN
  INIT_WORKING;
  COPY_QUERY_TO_WORKING_STORAGE(S, WS_SIZE);
  I := 1;
  Y := 0;
  WHILE (I <= WS_SIZE) DO
  BEGIN
    IF (WORKING_STORAGE[I, 1] <> 0) THEN
    BEGIN
      SET_FREQ;
      FREQ(S, L, WS_SIZE);
      MIN := 0.999;
      TEMP2 := 0;
      FIND_MIN_MAX(SK, L, MIN, TEMP2);
      Y := Y + 1;
      SELECTED_KEY[Y] := SK;
      FIND_COVERS(SK, S, WS_SIZE);
    END
    ELSE I := I + 1;
  END;
  TOTAL_RECORDS(SELECTED_KEY, Y, SUM);
  PRINT_OUT(SELECTED_KEY, Y, SUM, 70, 91);
END;

(* ------------------------ *)
PROCEDURE FIND_CONJUNCTS;
VAR
I: INTEGER;
BEGIN
WS_SIZE := 1;
FOR I := 1 TO Q_SIZE - 1 DO
  IF QUERY[I] = 0 THEN
    WS_SIZE := WS_SIZE + 1;
END;

(***************
(* PROCEDURE SET COVER *)
(**********************)
PROCEDURE SET_COVER;
VAR
I,J: INTEGER;
BEGIN
FOR J := 1 TO L DO
  FOR I := 1 TO WS_SIZE DO
    BEGIN
      COVER[I,J] := FALSE;
    END;
  FOR I := 1 TO WS_SIZE DO
    BEGIN
      KEYCOVER[I] := FALSE;
      S_KEYCOVER[I] := FALSE;
    END;
END;

(***************
(* PROCEDURE SORT *)
(**********************)
PROCEDURE SORT;
VAR
I,J,K,T: INTEGER;
BEGIN
  FOR I := 1 TO L DO
    BEGIN
      J := I;
      FOR K := J + 1 TO L DO
        IF (KEY_NOK[J] < KEY_NOK[K]) THEN
          BEGIN
            J := K;
            T := KEY_NOK[J];
            KEY_NOK[J] := KEY_NOK[K];
            KEY_NOK[K] := T;
          END;
    END;
END;

(***************
(* PROCEDURE MAKE KEY COPY *)
(**********************)
PROCEDURE MAKE_KEY_COPY;
VAR
I: INTEGER;
BEGIN
  FOR I := 1 TO L DO
    KEY_COPY[KEY_NO[I]] := I;
END;

(***************
(* PROCEDURE COPY QUERY TO COVER *)
(**********************)
PROCEDURE COPY_QUERY_TO_COVER;
VAR
I,J: INTEGER;
BEGIN
  S := 0;
  J := 1;
  FOR I := 1 TO Q_SIZE - 1 DO
    IF QUERY[I] = 0 THEN
      BEGIN
        S := S + 1;
        J := J + 1;
        FOR I := 1 TO WS_SIZE DO
          IF COVER[I,J] = TRUE THEN
            BEGIN
              J := J + 1;
              FOR I := 1 TO WS_SIZE DO
                COVER[I,J] := TRUE;
            END;
      END;
END;

(***************
(* PROCEDURE FIND POTENTIAL KEYS *)
(**********************)
PROCEDURE FIND_POTENTIAL_KEYS(VAR A: INTEGER);
VAR
I,J,K,T: INTEGER;
BEGIN
  COVER_FOUND := FALSE;
  I := 1;
  A := 1;
  WHILE NOT(COVER_FOUND) AND (I <= L) DO
    BEGIN
      Q COVER := TRUE;
      EXTRA := FALSE;
      FOR J := 1 TO WS_SIZE DO
        BEGIN
          TEMP := KEY_COVER[J];
          KEY_COVER[J] := KEY_COVER[J] OR COVER[I,J];
          BEGIN
            COVER[I,J] := COVER[I,J] OR COVER[I,J];
          END;
        END;
      END;
      BEGIN
        FOR J := 1 TO WS_SIZE DO
          IF COVER[I,J] = TRUE THEN
            BEGIN
              J := J + 1;
              FOR I := 1 TO WS_SIZE DO
                COVER[I,J] := TRUE;
            END;
      END;
    END;
  END;
END;
EXTRA := EXTRA OR (KEY_COVER[J] AND NOT(TEMP)) ;
END :
IF EXTRA THEN
BEGIN
SELECTED_KEY[A] := I :
A := A + 1 ;
END :
FOR M := 1 TO WS_SIZE DO
BEGIN
Q_COVER := Q_COVER AND KEY_COVER[M] :
END :
IF Q_COVER THEN
COVER_FOUND := TRUE ;
ELSE
I := I + 1 ;
END :

(******************************************************************************)
(= PROCEDURE ELIMINATE =)
(******************************************************************************)
PROCEDURE ELIMINATE( KEY_COVER : FCoverType ; VAR A : INTEGER ) ;
VAR
I,J,K,M : INTEGER ;
EXTRA : BOOLEAN ;
TEMP : BOOLEAN ;
BEGIN
WHILE NOT(MIN_COVER) DO
BEGIN
COVER_FOUND := FALSE ;
TEST_COVER := TRUE ;
I := 1 ;
WHILE NOT(COVER_FOUND) DO
BEGIN
EXTRA := FALSE ;
Q_COVER := TRUE ;
FOR J := 1 TO WS_SIZE DO
BEGIN
TEMP := KEY_COVER[J] ;
EXTRA := EXTRA OR (KEY_COVER[J] AND NOT TEMP) ;
END :
IF EXTRA THEN
BEGIN
SELECTED_KEY[B] := I ;
B := B + 1 ;
FOR M := 1 TO WS_SIZE DO
BEGIN
Q_COVER := Q_COVER AND KEY_COVER[M] :
END :
IF (Q_COVER) THEN
BEGIN
COVER_FOUND := TRUE ;
C := C + 1 ;
END :
>
PROCEDURE NEW_TECHNIQUE;
VAR
i, j: INTEGER;
BEGIN
FIND_CONJUNCTS;
SET_COVER;
MAKE_KEY_COPY;
COPY_QUERY_TO_COVER;
SORT;
MAKE_FREQSET;
FIND_POTENTIAL_KEYS(A);
MINIMUM=999;
MIN_COVER := FALSE;
TEST_COVER := TRUE;
FREQSET[F+1] := SELECTED_KEY[A-1];
FOR i := 1 TO F+1 DO
BEGIN
FOR i := 1 TO WS_SIZE DO
BEGIN
S_KEY_COVER[i] := COVER[FREQSET[i], i];
TEST_COVER := TEST_COVER AND S_KEY_COVER[i];
MIN_COVER := TEST_COVER;
END;
P.FINAL_KEYS[i] := FREQSET[i];
C := i;
IF NOT(MIN_COVER) THEN
ELIMINATE(S_KEY_COVER[i], A);
FOR i := 1 TO C-DO
P.FINAL_KEYS[i] := KEY_NOCP_FINAL_KEYS[i];
TOTAL_RECORDS(P.FINAL_KEYS, C, SUM);
END;
END;
END;
Appendix C

Results
<table>
<thead>
<tr>
<th>PRIME KEY WORD TECHNIQUE</th>
<th>MINIMUM/MAXIMUM TECHNIQUE</th>
<th>COMBINATION TECHNIQUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS FOR RETRIEVAL</td>
<td>RECORDS 8</td>
<td>KEYS FOR RETRIEVAL</td>
</tr>
<tr>
<td>K3 K5</td>
<td>5</td>
<td>K5</td>
</tr>
<tr>
<td>K2 K5 K3 K9 K7</td>
<td>26</td>
<td>K2 K3 K9 K5 K7</td>
</tr>
<tr>
<td>K9 K14 K1</td>
<td>24</td>
<td>K1 K9 K5 K14</td>
</tr>
<tr>
<td>K8</td>
<td>8</td>
<td>K3 K8</td>
</tr>
<tr>
<td>K5 K16</td>
<td>21</td>
<td>K5 K16</td>
</tr>
<tr>
<td>K4 K8</td>
<td>12</td>
<td>K4 K8</td>
</tr>
<tr>
<td>K4</td>
<td>4</td>
<td>K4</td>
</tr>
<tr>
<td>K9</td>
<td>9</td>
<td>K9</td>
</tr>
<tr>
<td>K7</td>
<td>7</td>
<td>K7</td>
</tr>
<tr>
<td>K7</td>
<td>7</td>
<td>K7</td>
</tr>
<tr>
<td>K8</td>
<td>8</td>
<td>K8</td>
</tr>
<tr>
<td>K3 K7 K9 K8</td>
<td>27</td>
<td>K9 K12</td>
</tr>
<tr>
<td>K6 K17</td>
<td>6</td>
<td>K6</td>
</tr>
<tr>
<td>K6</td>
<td>6</td>
<td>K6</td>
</tr>
<tr>
<td>K10</td>
<td>10</td>
<td>K10</td>
</tr>
<tr>
<td>K3 K5 K8</td>
<td>16</td>
<td>K5 K3 K8</td>
</tr>
<tr>
<td>K2 K8</td>
<td>10</td>
<td>K2 K8</td>
</tr>
<tr>
<td>K5</td>
<td>5</td>
<td>K5</td>
</tr>
<tr>
<td>K5 K4</td>
<td>9</td>
<td>K5 K4</td>
</tr>
<tr>
<td>K7</td>
<td>7</td>
<td>K7</td>
</tr>
<tr>
<td>K7</td>
<td>7</td>
<td>K7</td>
</tr>
<tr>
<td>K7 K2 K10 K9</td>
<td>28</td>
<td>K2 K7 K9 K10</td>
</tr>
<tr>
<td>K8</td>
<td>8</td>
<td>K2 K8</td>
</tr>
<tr>
<td>K3</td>
<td>3</td>
<td>K3</td>
</tr>
<tr>
<td>K10 K7 K17 K9 V(K11 K5 K13 V(K3 K7) V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k11-k10-k12) v (k11-k12-k14-k15-k9)</td>
<td>* k10 k9</td>
<td></td>
</tr>
<tr>
<td>(k10-k9-k12) v (k7-k13-k12-k14) v (k14-k7-k10) v (k11-k12-k10) v (k12-k10-k12)</td>
<td>* k9 k7 k8 k10</td>
<td></td>
</tr>
<tr>
<td>(k11-k10) v (k2-k12-k14-k7-k5) v (k7-k12-k10-k13-k16) v (k6-k15-k8-k7) v (k13-k7-k9)</td>
<td>* k1 k2 k3 k10</td>
<td></td>
</tr>
<tr>
<td>(k14-k9) v (k9-k7)</td>
<td>* k7 k3</td>
<td></td>
</tr>
<tr>
<td>(k12-k14-k8-k7-k10) v (k5-k3) v (k9-k7) v (k11-k9-k8-k5)</td>
<td>* k7 k3</td>
<td></td>
</tr>
<tr>
<td>(k8-k6-k5)</td>
<td>* k5</td>
<td></td>
</tr>
<tr>
<td>(k10-k9) v (k9-k2) v (k9-k14-k5)</td>
<td>* k9</td>
<td></td>
</tr>
<tr>
<td>(k8-k4) v (k8-k10-k9)</td>
<td>* k4 k8</td>
<td></td>
</tr>
<tr>
<td>(k9-k16-k13)</td>
<td>* k9</td>
<td></td>
</tr>
<tr>
<td>(k14-k3-k4-k18-k15) v (k7-k9) v (k3-k11-k4-k6-k10) v (k17-k10-k15) v (k10-k12-k13)</td>
<td>* k3 k7 k4 k10</td>
<td></td>
</tr>
<tr>
<td>(k9-k3-k11-k5) v (k12-k9-k7-k8)</td>
<td>* k3 k7</td>
<td></td>
</tr>
<tr>
<td>(k5-k3-k7) v (k12-k14) v (k8-k10-k14)</td>
<td>* k3 k12 k8</td>
<td></td>
</tr>
<tr>
<td>(k8-k15-k11) v (k6-k8-k10) v (k9-k16-k4) v (k11-k9) v (k12-k3-k17-k13) v (k15-k16-k4)</td>
<td>* k8 k4</td>
<td></td>
</tr>
<tr>
<td>(k10-k8) v (k15-k11) v (k14-k15) v (k9-k10) v (k13-k4-k3) v (k14-k7-k11-k9-k7) v (k9-k10) v (k12-k10)</td>
<td>* k8 k11 k14 k3 k9</td>
<td></td>
</tr>
<tr>
<td>(k10-k11) v (k6-k17-k9-k8-k16)</td>
<td>* k10 k6</td>
<td></td>
</tr>
<tr>
<td>(k6-k11-k7) v (k13-k16-k15-k6)</td>
<td>* k6</td>
<td></td>
</tr>
<tr>
<td>(k10-k8-k6-k9-k11)</td>
<td>* k6</td>
<td></td>
</tr>
<tr>
<td>(k6-k11)</td>
<td>* k11</td>
<td></td>
</tr>
<tr>
<td>(k13-k8-k5-k18) v (k7-k5)</td>
<td>* k5</td>
<td></td>
</tr>
<tr>
<td>(k13-k9-k10) v (k16-k10-k17-k8-k11)</td>
<td>* k9 k8</td>
<td></td>
</tr>
<tr>
<td>(k9-k13) v (k8-k10-k13-k4) v (k6-k4-k7) v (k17-k10-k14) v (k10-k18-k13-k12) v (k5-k8-k13-k10) v (k13-k9-k10-k9-k4)</td>
<td>* k0 k4 k10</td>
<td></td>
</tr>
<tr>
<td>(k11-k7-k6-k9) v (k10-k11-k10-k14) v (k7-k13-k10-k12)</td>
<td>* k6 k10</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>K5 K9</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>K5 K9</td>
<td>13</td>
</tr>
<tr>
<td>25</td>
<td>K6 K9 K7</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>K12 K2</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>K6 K7</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>K6</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>K11</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>K14 K4 K1</td>
<td>27</td>
</tr>
</tbody>
</table>

**END OF REPORT**
BIBLIOGRAPHY


Burkhard, Walter A. "Associative Retrieval Trie Hash-Coding."


