INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the original text directly from the copy submitted. Thus, some dissertation copies are in typewriter face, while others may be from a computer printer.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyrighted material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is available as one exposure on a standard 35 mm slide or as a 17" × 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. 35 mm slides or 6" × 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
Accessing the World's Information since 1936
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
Essays on some critical issues in physical database design

Park, June Sung, Ph.D.
The Ohio State University, 1988
ESSAYS ON SOME CRITICAL ISSUES IN PHYSICAL DATABASE DESIGN

D I S S E R T A T I O N

Presented in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in the Graduate School of the Ohio State University

By

June Sung Park, B.B.A., M.B.A.

***

The Ohio State University

1988

Dissertation Committee:       Approved by
 Jon Cunnyngham
 Yao-Nan Lien

One-of-a-Kind Ph. D. Program in Information Systems
VITA

November 10, 1954  Born - Seoul, Korea

1978-1980  Planning Staff, Korea Development Finance
          Corporation, Seoul, Korea

1979  B.B.A., Seoul National University, Seoul, Korea

1980-1982  Research Staff, Korea Institute for Industrial
           Economics and Technology, Seoul, Korea

1982-1983  Research Fellow, Institute of Economic Research,
           Hanyang University, Seoul, Korea

1983  M.B.A., Seoul National University, Seoul, Korea

1984-1987  Graduate Research Associate, Computing Services
           Center, College of Business, The Ohio State
           University, Columbus, Ohio

1987-Present  Tenure Track Professor, Department of
              Quantitative Business Analysis, Louisiana State
              University, Baton Rouge, Louisiana

PUBLICATIONS

An integrated model of record segmentation and access path selection for

Optimal reorganization policies for stationary and evolutionary
databases. (with R. Bartoszynski, P. De and H. Pirkul), Being revised
for resubmission to Management Science.

A rule-based query optimizer for vertically segmented relational
databases. (with S. Shen), Submitted for journal publication.

An adaptive heuristic for file migration in local area networks. (with

Policies for the Promotion of New Technology-Based Small Firms in Korea
(with C. Kim and W. Yeo), Korea Science Foundation, August 1983.

Protectionism in Developed Countries and World Textile Industry, Korea International Economic Institute, July 1981.

FIELDS OF STUDY

Major Field: Information Systems

Studies in Computer and Information Systems, Industrial and Systems Engineering, Statistics and Finance
# TABLE OF CONTENTS

VITA ............................................................ ii

LIST OF TABLES .................................................... viii

LIST OF FIGURES .................................................... ix

INTRODUCTION .................................................... 1

CHAPTER PAGE

PART I  AN INTEGRATED MODEL OF RECORD SEGMENTATION AND ACCESS PATH SELECTION FOR DATABASES

I.1 Introduction ........................................... 5

I.2 Record Segmentation and Access Path Selection ............ 10

I.3 The Design Process ..................................... 13

I.4 Problem Definition ....................................... 16

I.4.1. The paradigm of record segmentation ............ 16

I.4.2. Structure of the database ...................... 17

I.4.3. Operations on subfiles ......................... 20

I.5 Model Formulation ..................................... 26

I.5.1. A generic framework ............................. 26

I.5.2. The cost model .................................... 29

I.5.2.1. Function G₁ .................................... 30

I.5.2.2. Function G₂ .................................... 31

I.5.2.3. Function G₃ .................................... 33

I.5.2.4. Function G₄ .................................... 34

I.5.2.5. Total cost function ......................... 35

I.6 Solution Procedures ..................................... 38

I.6.1. Heuristic solution procedure RSEGH ............... 39

I.6.2. Optimal solution algorithm ...................... 43

I.6.2.1. Decomposition of (NLP) ...................... 43
1.6.2.2. Strategies for fathoming (ILP) by bounds 46

1.6.2.2.1. Linear programming relaxation of (ILP) 47

1.6.2.2.2. Lagrangian relaxation of (ILP) 48

1.6.2.3. Final algorithm RSEG 51

1.7 Computational Results 55

1.8 Conclusions 63

PART II A RULE-BASED QUERY OPTIMIZER FOR VERTICALLY SEGMENTED RELATIONAL DATABASES

II.1 Introduction 65

II.2 The Model DBMS and Access Path Structure 68

II.3 Query Parameters 70

II.4 Selection of an Access Method 71

II.5 Selection of Access Paths 85

II.5.1. Search strategies for processing r(I) 85

II.5.2. Search strategies for processing r(II) 88

II.6 Query Optimization 91

II.6.1. Optimal subfile access sequence 93

II.6.2. Processing subfiles 98

II.7 Example 102

II.8 Conclusions 104

PART III OPTIMAL REORGANIZATION POLICIES FOR STATIONARY AND EVOLUTIONARY DATABASES

III.1 Introduction 106

III.1.1. The literature 107

III.1.2. The approach 109

III.2 A Theory of Optimal Reorganization Policies 113

III.2.1. Basic assumptions on the file structure and operations 113
III.2.2. Model formulation ............................ 115
III.2.3. Solution procedures .......................... 122
  III.2.3.1. Evolutionary file system ............ 122
  III.2.3.2. Stationary file system ............ 126

III.3 Applications of the Theory ........................... 133
  III.3.1. A stochastic model of file state dynamics . 133
  III.3.2. An analytic model of the file structure:
         the ISAM case ................................... 135
           III.3.2.1. Parameterization of the file
                       structure ............................. 135
           III.3.2.2. Overflow and capacity exhaustion
                       problems ............................ 138
           III.3.2.3. Cost functions ....................... 140

III.4 Conclusions ...........................................144

PART IV  AN ADAPTIVE HEURISTIC FOR FILE MIGRATIONS IN LOCAL AREA
          NETWORKS

 IV.1 Introduction .............................................147
 IV.2 System Description .................................... 151
 IV.3 File Access Pattern and Decision Structure ............ 155
 IV.4 System Costs and the Optimization Model ............ 160
 IV.5 Adaptive File Migration Policy ......................... 163
   IV.5.1. Adaptive policy under no storage capacity
           constraint ....................................... 163
   IV.5.2. Adaptive policy under storage capacity
           constraint ....................................... 166

 IV.6 Conclusions ..........................................171

APPENDICES

   A. Proof of Theorem 1 ...................................... 172
   B. Proof of Corollary 1 .................................... 174
   C. Proof of Theorem 2 .................................... 175
   D. Subroutine SUBAL1 ...................................... 176
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Parameters of a Relation Descriptor</td>
<td>19</td>
</tr>
<tr>
<td>1.2 Conceptual-Level Categories of Single Key Queries</td>
<td>22</td>
</tr>
<tr>
<td>1.3 Parameters of a Query Descriptor</td>
<td>23</td>
</tr>
<tr>
<td>1.4 Internal-Level Classification of Operations</td>
<td>25</td>
</tr>
<tr>
<td>1.5 Storage Device Parameters</td>
<td>29</td>
</tr>
<tr>
<td>1.6 Taxonomy of Problem Instances</td>
<td>56</td>
</tr>
<tr>
<td>1.7 Analysis on the Gap and the Expected Run Time of Heuristic Solutions</td>
<td>57</td>
</tr>
<tr>
<td>1.8 Analysis on the Average Run Time of Algorithm RSEG</td>
<td>60</td>
</tr>
<tr>
<td>1.9 Percentage Cost Savings by the Optimal Record Segmentation</td>
<td>61</td>
</tr>
<tr>
<td>2.1 The Trigger Level of the Selectivity: Eq. (2.7)</td>
<td>82</td>
</tr>
<tr>
<td>2.2 The Trigger Level of the Selectivity: Eq. (2.9)</td>
<td>83</td>
</tr>
<tr>
<td>2.3 Rules for Query Optimization</td>
<td>94</td>
</tr>
<tr>
<td>2.4 Notations for the Sizes of the Relational Algebras</td>
<td>97</td>
</tr>
<tr>
<td>3.1 File Structure and Storage Device Parameters</td>
<td>136</td>
</tr>
<tr>
<td>4.1 The Contribution of File i to the System Cost $T_{C_n}$</td>
<td>164</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Optimal Values, (LP) Bounds and (LR) Bounds of (ILP)$p$ on the Domain L</td>
<td>49</td>
</tr>
<tr>
<td>2.1 The Trigger Level of the Selectivity: Eq. (2.7)</td>
<td>80</td>
</tr>
<tr>
<td>2.2 The Trigger Level of the Selectivity: Eq. (2.9)</td>
<td>81</td>
</tr>
<tr>
<td>2.3 The Correspondence between Views and Rules in Query Optimization</td>
<td>90</td>
</tr>
</tbody>
</table>
INTRODUCTION

The dissertation investigates four important issues in physical database design and maintenance. In Part I of this dissertation, an analytic model is developed to integrate two closely related subproblems of physical database design: vertical segmentation and access path selection. Several restrictive assumptions of the past research on record segmentation, e.g., a single access method and the dominance of one subfile over the other, are relaxed in this model. A generic design process for this integrated performance model is suggested and applied to a relational database. A heuristic procedure and an optimal algorithm are developed for solving the model. Extensive computational results are reported to show the effectiveness of these solution techniques.

Vertical segmentation is shown to be an effective database structuring technique in Part I, which can save database operating costs by 30-60%. In Part II, we develop a low-level optimizer for queries in conjunctive normal form which are processed on vertically segmented relations. The optimizer takes into account the presence of both clustered and non-clustered indexes, and incorporates such factors as the optimal subfile access sequence and the optimal selection of an access method based on the size of a relational algebra applied. The optimizer is formalized as a rule-based system, and a prototype implementation module coded in LISP is provided in Appendix E.
In Part III, the problem of determining optimal reorganization policies for databases which employ file structures with overflow chaining is studied. The dynamics of database performance driven by update transactions and reorganizations is formulated as a stochastic control model which incorporates micro-level design parameters of the physical file structure. Various restrictive assumptions employed in the past research, e.g., the linear performance deterioration, the ergodicity of state dynamics or the deterministic growth of the file size, are relaxed in this Part. Polynomial-time procedures for solving the optimization models are developed for two cases: when the file size is stationary as in the steady-state and when the file size stochastically evolves with a nonlinear trajectory. Applications of the models and the solution procedures to real-life databases are illustrated by examples.

In Part IV, the file migration problem in distributed database systems is studied. We develop an adaptive file migration algorithm for LAN-based DCSs based on the observed behavior of users' file request pattern. A multi-access broadcast communication network is assumed and distributed file migration decisions at individual sites are assumed allowing the site autonomy. File migration decisions are considered to be made at fixed short time intervals and Bayesian approach is used for adaptive forecasting of file access rates over the next decision interval. We develop a dynamic optimization model for the file migration policy using Bellman approach. Based on this model, we derive a heuristic algorithm which can be used in multi-period independent migration decisions. It will be shown that this heuristic produces
solutions almost identical to those of dynamic backward induction, if there is contention among remotely referenced files for the local storage space. We also investigate the effect of local storage expansion on the network traffic load.
PART I

AN INTEGRATED MODEL OF RECORD SEGMENTATION AND
ACCESS PATH SELECTION FOR DATABASES
I.1 Introduction

The objective of physical database design is to determine how to physically organize the logical database in secondary memory (conceptual-internal mapping of the database) so that the information requirements of the user community can be met most efficiently. Record partitioning, an important subproblem of this physical design, deals with the assignment of attributes to multiple record segments so that the overall performance of the database is optimized. The special case of record partitioning when at most two record segments are considered is known as the record segmentation problem.

Database performance measured in terms of the processing time of user requests is critically affected by the strategy chosen for searching the response set, a set of tuples that satisfies a query. The search strategy, or access path selection, is implemented in two stages. The configuration of the access path structures is implemented in the physical database design stage, and the query optimization is performed when each user request is processed. The past research on record partitioning [12,47,64,68,71,88,89,92,91,126] has by and large ignored the issue of access path selection.
A heuristic solution to the record partitioning problem is developed in [68] and [71]. The number of record segments is not restricted and determined endogenously within the solution procedure. Under the assumption that all the retrieval queries and updates are processed sequentially, a pairwise similarity measure is designed among attributes based on a co-access probability. A mathematical clustering algorithm [95] is used to form initial clusters of attributes based on those similarity measures. Since the initial clusters are generated usually with fuzzy boundaries, human judgement is involved to establish the initial clusters. An efficient regrouping of initial clusters is determined by a branch-and-bound algorithm with a cost model in low-level details as the objective function.

A mathematical programming approach to the record segmentation problem is proposed in [47], [88] and [92]. The relation is partitioned into a primary subfile and a secondary subfile, and only retrieval queries are considered. All query processings must first access the primary subfile, even though the query does not request any attribute in that subfile. The secondary subfile is accessed if it turns out, after searching the primary subfile, that the query cannot be satisfied by the primary subfile alone. This restriction (dominance of one subfile over the other) leads to additional search time, but is imposed to make the mathematical formulation tractable [126]. It is assumed that either all the queries are sequentially processed, or all are processed randomly via secondary indexes. For random access, it is also assumed that both subfiles are unblocked. Such simplifying assumptions on access path design and file structure are employed also to make the model tractable.
Under these assumptions, efficient solution procedures based on [53] and [59] are developed, guaranteeing optimality in polynomial time.

As in [68] and [71], a heuristic clustering procedure to determine an efficient partitioning is developed also in [64]. This approach tackles random access to the blocked secondary memory, assuming only retrieval queries. The cost model estimates the expected number of block accesses [137] ignoring the hardware complexities of a movable-head disk environment. The hill climbing approach employed in the clustering procedure is subject to cycles and convergence to a local optimum. The quality of the heuristic solutions in [64,68,71] has not been established, because no analysis has been presented showing the gaps of these solutions from theoretical bounds on the optimal values. The cost reductions realized by these solutions compared with the unsegmented relation or the trivial partitioning, with each attribute assigned to a separate subfile, have been reported [64,89]. However, such a relative measure of cost reduction can not be viewed as a criterion for evaluating the quality of solution, because the cost reduction realizable by record partitioning is highly dependent on database structures and the pattern of information requests of the user community.

In this Part we present an integrated model of record segmentation and access path selection. The paradigm of record segmentation employed in this Part is different from the past research [47,88,92] in that we eliminate the assumption of dominance of a primary subfile over a secondary subfile. As a result, there is no need to
access a subfile that has no attribute involved in a query. The cost
model reflects reasonably low-level details of hardware architecture.
Formally incorporating both sequential and random access paths, the cost
model is formulated as a 0-1 nonlinear program, which is NP-hard [103].
We present an efficient heuristic procedure as well as an optimal
algorithm for its solution, and report statistical evidence for the
quality of the heuristic solution by performing experiments with a wide
variety of problem instances.

The model formulated in this Part, however, shares some
restrictions of the past research. It assumes only unary operations,
i.e., selections and projections from a single relation. We also limit,
for convenience, the scope of user requests to include only retrieval
queries. We, however, suggest a general design process and a generic
framework for the cost model of record segmentation which is applicable
in more realistic situations than are assumed in this Part. This
framework facilitates easier understanding of the underlying mechanisms
for optimizing record segmentation, integrated with other issues in
physical database design such as access path structuring and query
optimization. The basic notion of the design process is to classify user
requests into internally-equivalent groups by analyzing the access paths
used for individual requests. Such conceptual-internal mapping of user
requests in the design process makes the model formulation modularized,
and makes the current model grow incrementally without significant
structural changes as some of the current assumptions on the scope of
user requests are relaxed.
Chapter 1.2 of this Part addresses the problems in integrating record segmentation with access path selection. In Chapter 1.3, we present a general process for an integrated design of record segmentation and access path selection, which is utilized in the rest of the Part. In Chapter 1.4, we suggest a new paradigm of record segmentation, and elaborate our assumptions on the internal-level database structures and the operations to be performed on the database. In Chapter 1.5, a generic framework for optimizing the integrated model, as well as a detailed cost model under the assumptions described in the previous chapter are developed. In Chapter 1.6, we present a heuristic algorithm as well as an optimal solution procedure, and in Chapter 1.7, we discuss results of experiments with these procedures. Chapter I.8 provides some concluding remarks.
1.2 Record Segmentation and Access Path Selection

In this chapter we address the problems encountered when one tries to integrate record segmentation with access path selection. We omit any discussion on the significance of record segmentation per se. The reader is referred to [47, 88, 89, 92, 126] for a description of how record segmentation improves database performance.

The extension of the approach employed in [47, 88, 92] to incorporate random accesses with blocked secondary memory is limited by its solution procedure. The expected number of block accesses required for random processing of a query [19, 133, 137] is expressed as a complex nonlinear function of the decision variables. Such nonlinearity introduced into the model structure will require a solution procedure totally different from that of [47, 88, 92]. Therefore, the heuristic algorithm in [64] has been applied for solving the case in which random accesses with blocked secondary memory are considered [91].

What is needed is a comprehensive model and a solution procedure that can optimize the segmentation of a relation for which any of the access paths, i.e. sequential, random via primary key, or random via secondary key, can be selected for query processing. The model should also work in a blocked (or paged) secondary memory environment, since
most real-life databases operate in that environment. The record segmentation decision becomes more complicated when each user request is processed by a choice from alternative access paths. The source of complexity that comes with the combined consideration of different access methods is the close interaction between record segmentation and access path selection in the deep level of implementation. The efficiency of sequential accesses to the response set of a query is significantly enhanced by record segmentation. Record segmentation and random access, however, have an offsetting effect on each other with respect to the performance of the database. Either one can contribute significantly to database performance, but can greatly reduce the contribution of the other [89]; however, both combined can still realize more improvements in performance than either alone under a blocked secondary memory environment.

Suppose, as an extreme case, that the tuples in the response set of a query are widely dispersed across a subfile so that at most one tuple from the response set is contained in a block, and the storage device is paged with a fixed blocksize. Then, the number of page accesses and the proportion of relevant information per page in processing a query will be the same under any segmentation of a relation as under no segmentation. In other words, no cost savings will be achieved by record segmentation in this situation. Usually, however, when the selectivity of the response set exceeds a certain level, there exists some difference in the expected number of block accesses between an unsegmented relation and segmented subfiles with reduced tuple lengths, even when random access is employed. As the selectivity
increases, the difference between the expected number of tuples in the response set found in a block of a subfile and that found in a block of an unsegmented relation widens. This difference also widens if the tuple length of a subfile becomes shorter. As long as there exists such differences, the efficiency of retrieving the desired tuples via a random access path can be improved by record segmentation, although the degree of such improvement may still be significantly less than that for a sequential access.
I.3 The Design Process

In this chapter we present a generic design process for record segmentation in conjunction with access path selection. This design process follows the unifying approach to physical database design proposed in [12]. Each subfile is assumed to be stored as a simple file or a base table. Therefore, the modeling of record segmentation will encompass: (i) a description of the physical structure of subfiles and access paths, (ii) an identification of operations performed on subfiles based on the access path selection, and (iii) a development of cost expressions for those operations. The objective is to minimize the cost function formulated in this way.

We assume that the segmentation is designed for each single relation independently from other relations in the database. This assumption is valid if a single relation comprises a database, or no binary operations are employed by the user in a multi-relation database. The design process has five steps:

Step 1: Define the paradigm of record segmentation.

Step 2: Identify and model the physical structure of the database.

A description of the physical database structure in the context
of record segmentation includes the structure of each subfile, primary and secondary access paths to the relation, and the linkage between the two subfiles. In principle, file structures of the two subfiles may differ.

Step 3: Identify the operations to be performed on the database, based on the record segmentation scheme defined in Step 1 and the model of the physical database structure abstracted in Step 2.

In this step, we first identify all the user requests that are expected to be encountered during normal processing, and then map those user requests into internally-equivalent groups of operations based on the selected query optimization algorithm. A subfile will be the basic unit of storage on which we identify the operations. We need the following definition for this step.

Definition 1 (Operation Type): A set of user requests is said to follow the same "access rule", and mapped into one "operation type", if they have (i) the same criteria for determining the subfile(s) to be accessed and the order in which the subfiles are accessed, (ii) the same access path for searching the addresses of tuples (in the subfile first accessed) that are requested by the user, and (iii) the same access method for the subfile secondly accessed.

According to this definition, a large set of user requests is typically mapped to a small set of internally-equivalent operation types, by analyzing the access paths to the subfiles that are employed in processing individual requests. The mechanisms of such conceptual -
internal mapping of the user requests will become clear when its application is shown in Chapter 1.4.

Step 4: Formulate the cost expressions for each of the operation types identified in Step 3.

A generic framework for the cost model can be used for this step. The detailed cost model, however, depends on the assumptions made in Step 1 through Step 3. The relevant information about the logical and physical database structures, user requests and storage device is incorporated into the cost model as input parameters. The overall cost is the sum of cost expressions for different operation types.

Step 5: Develop an efficient solution algorithm for the cost model formulated in Step 4.

The decision variables in the record segmentation problem are 0-1 variables. The cost structure becomes a complex nonlinear function when random access to a blocked secondary memory is incorporated into the model. The cost model is, therefore, formulated as a 0-1 nonlinear program which is NP-hard [103]. Thus, for realistic-size problems, instead of looking for true optimality, it may be practical to devise an efficient heuristic procedure that provides near-optimal solutions.
I.4 Problem Definition

I.4.1. The paradigm of record segmentation

The past research on record segmentation has been motivated by an assumed characteristic of commercial databases, namely the "80/20" rule of thumb [77,92,126]. The paradigm of record segmentation proposed in [47,88,92], which assumes two distinct types of subfiles - primary and secondary, is tailored to this 80/20 rule. The performance of this paradigm, however, deteriorates considerably, if the database usage pattern does not follow the 80/20 rule. In such a situation, the cost of accessing the primary subfile without retrieving any attribute could prove to be totally undesirable. The paradigm of record segmentation employed in this Part is indifferent to the 80/20 rule. It does not incur the unnecessary cost of accessing a subfile which does not have any requested attribute, and, therefore, is applicable to any database no matter what the usage pattern is. We now present a formal definition of record segmentation.

Definition 2 (Record Segmentation): If a relation R has a schema \(d_1d_2...d_m\) for some fixed arity m of R, we define the record segmentation as vertically partitioning R into two sub-relations (subfiles) \(R_1\) and \(R_2\). 
having schemata $\delta_1 \delta_2 \ldots \delta_i$ and $\delta_1 \delta_2 \ldots \delta_j$, respectively, where the sets of attributes, $D = \{d_1, d_2, \ldots, d_m\}$, $D_1 = \{\delta_1, \delta_2, \ldots, \delta_i\}$, and $D_2 = \{\delta_1, \delta_2, \ldots, \delta_j\}$, are such that $D_1 \cup D_2 = D$ and $D_1 \cap D_2 = \phi$.

We call the tuples of $R_1$ and $R_2$ record segments. Conversely, a subfile is defined to be a set of mappings from the names of attributes assigned to a record segment to their values. This definition implies that the order in which the attributes are assigned in a record segment does not matter in our analysis. We will use the terms record segment and tuple interchangeably.

It is assumed that a list of attributes in each record segment is kept in the main memory. Given a user request, the database management system (DBMS) decides which of the two subfiles to access. The DBMS accesses only the subfile(s) which contain(s) the attributes to be used to satisfy a given user request. This "subfile access rule" distinguishes our approach from the past research [47,88,92].

I.4.2. Structure of the database

The storage system of a base table is assumed to be that of the relational database proposed in [5,15]. Thus, a fixed-length page is a unit of transfer between main memory and secondary memory, and a tuple in each subfile is identified by a tuple identifier (TID) which has two components: page number and offset within a page.
Assumption 1 (Linkage between Subfiles): Record segments of the same record occurrence have the same relative tuple number in their respective subfiles. The DBMS maintains an address map in main memory that matches a relative tuple number with a corresponding TID in each subfile [89].

Assumption 1 eliminates the need for explicit storage for linkage between two subfiles, and implies that once a tuple in one subfile is located, then the corresponding tuple of the same record occurrence in the other subfile can be accessed directly based on the TID.

The primary and secondary access path structures are assumed as follows:

Assumption 2 (Primary Access Path): A multi-level static directory structured as a B+ tree supports both sequential and random accesses based on primary key values. Relative tuple numbers in each subfile correspond to the increasing order of the primary key value. Each node in the index set of the B+ tree, other than the root which is stored in main memory, is stored as a block in the secondary memory where the blocksize is variable.

Assumption 3 (Secondary Access Path): An inverted file directory supports random access based on secondary key values. The physical layout for the inverted file organization is the same as that proposed in [19].
The assumption of varying block sizes for the primary directory is required as a precondition for the validity of Assumption 6 in Chapter 1.5. We describe here only some features of the inverted file organization that are relevant for later analyses (See [19] for details): (i) all indexes are blocked with the same block size, (ii) index entries in the key-value index area are ordered first by key-name and then by key-value, and the address list is ordered by the value of address, and (iii) the addresses in the address list area are TID's. Either subfile can be designated as the one to be inverted because of Assumption 1.

Table 1.1. Parameters of a Relation Descriptor

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_i )</td>
<td>fixed length in bytes of an attribute ( d_i ), ( i = 1, 2, \ldots, m )</td>
</tr>
<tr>
<td>( m )</td>
<td>arity of the relation</td>
</tr>
<tr>
<td>( N )</td>
<td>number of tuples in the relation</td>
</tr>
<tr>
<td>PK</td>
<td>a singleton set of primary key attribute*</td>
</tr>
<tr>
<td>SK</td>
<td>a set of secondary key attributes*</td>
</tr>
<tr>
<td>NK</td>
<td>a set of non-key attributes*</td>
</tr>
</tbody>
</table>

* Attributes are expressed as column numbers in the relation.
Attributes of a relation are classified into three categories: primary key, secondary key, and non-key. Non-keys denote attributes other than the primary and secondary keys. We assume for convenience that the relation has a single attribute for the primary key. It is also assumed that the selection of the attributes to be inverted is determined exogenously prior to the determination of record segmentation. A relation is characterized by a collection of parameters such as those in Table 1.1.

1.4.3. Operations on subfiles

A query is interpreted as a set of transactions having the same content, i.e., the same projection schema and the same selection formula. We assume that the frequency \( f_j \) of a query \( Q_j \) during the planning period is known with certainty. Using the relational algebra, \( Q_j \) is expressed as \( \pi_{p_j} \sigma_{F_j}(R) \), where \( \pi \) is a projection operator, \( \sigma \) is a selection operator, \( p_j \) is a projection schema (a set of attributes required to be displayed), and \( F_j \) is a selection formula.

We classify queries into two broad categories according to the selection formula \( F \): (i) "single key queries" with \( F = \bigwedge_{k=1}^{\omega} (d_i \theta v_k) \), i.e., a disjunction of \( \omega \) atomic predicates which relate the same attribute \( d_i \) to different values \( v_k \), \( k=1, \ldots, \omega \); and (ii) "multiple key queries" in disjunctive normal form with the selection formula referencing more than one attribute. The single key query is further classified: if \( \omega=1 \), i.e., if a query has only a single atomic predicate,
we call it the "atomic query"; otherwise (when $\omega>1$), queries are called
the "single key disjunctive query." An atomic predicate is called the
"simple" atomic predicate when the relational operator $\theta$ is $=$, and the
"range" atomic predicate when it is $<$, $>$, $\leq$, or $\geq$. Atomic predicates
with the $\neq$ operator will be ignored without loss of generality. (1)

Assumption 4 (Scope of User Requests): All user requests under
consideration are single key queries. (Hereafter a user request is
simply called a query.)

Assumption 4 excludes update operations and binary operations
such as Cartesian product, join, natural join, etc. For simplicity of
analysis, multiple key queries are excluded in general. It will be
shown, however, in Chapter I.5, that the model and solutions presented
in this Part readily subsume the special case of multiple key queries
where the attributes in the selection formula are all inverted for
random access. Every single key query falls into one of the categories
classified in Table 1.2. We assume for convenience that the expected
number $r_j$ of tuples to be selected for a query $Q_j$ is known. A query is
categorized by a collection of parameters as shown in Table 1.3.

We employ the following simple algorithm for optimizing
selections for single key queries.

Assumption 5 (Query Optimization Algorithm): (i) If $S_j \in PK$, then use
the primary index to find physical addresses for the response set. (ii)
If $S_j \in SK$ and $r_j \leq \hat{r}$, i.e., the cardinality of the response set of the
A query is smaller than a certain level, r (hereafter called a trigger level), then use the secondary index to find physical addresses for the response set.

Table 1.2. Conceptual-Level Categories of Single Key Queries

<table>
<thead>
<tr>
<th>Category</th>
<th>Retrieval without selection formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category b: Retrieval with selection formula</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>selection attribute</th>
<th>selection formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>(subcategory of b)</td>
<td>atomic</td>
</tr>
<tr>
<td>primary key(b.1)</td>
<td>b.1.1</td>
</tr>
<tr>
<td>secondary key(b.2)</td>
<td>b.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>non-key(b.3)</td>
<td>b.3.1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* On an internal level, each atomic predicate in the disjunctive selection formula can be implemented as a separate atomic query without any change in processing cost.
(iii) If $S_j = \phi$ (no selection formula), or $S_j \in \text{NK}$, or $S_j \in \text{SK}$ and $r_j > \hat{r}$, then read all the tuples of the subfile containing the selection attribute.

If the cardinality of the response set of a query is considerably large, a sequential search of the whole subfile can be faster than directly accessing the desired tuples [118]. Selectivity of more than 10 to 20 percent often justifies a sequential search of the whole file [20]. Therefore, only those attributes whose image size [128] would yield an average selectivity of less than 10-20% would qualify for inversion. Since an optimal selection of attributes for inversion is assumed to precede the record segmentation decision, it is most likely that only the single key disjunctive queries will exceed the trigger level as far as secondary key selections are concerned.

Table 1.3. Parameters of a Query Descriptor

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>number of different queries transacted over the planning period</td>
</tr>
<tr>
<td>$f_j$</td>
<td>frequency of a query $Q_j$ over the planning period, $j = 1, 2, ..., n$</td>
</tr>
<tr>
<td>$S_j$</td>
<td>an attribute* that appears in the selection formula of $Q_j$</td>
</tr>
<tr>
<td>$P_j$</td>
<td>a set of attributes* that are in the projection schema of $Q_j$</td>
</tr>
<tr>
<td>$r_j$</td>
<td>cardinality of the response set of $Q_j$</td>
</tr>
</tbody>
</table>

* Attributes are expressed as column numbers in the relation.
Now, we map the nine categories of queries at the conceptual level (Table 1.2) to the operation types at the internal level of the database following Definition 1. We identify four different operation types $U_c$, $t=1,2,3,4$, based on the query optimization algorithm for selections as well as the access rules for projections: If $S_j \in \text{PK}$, then $Q_j \in U_1$; if $S_j \in \text{SK}$ and $r_j \leq \hat{r}$, then $Q_j \in U_2$; if $S_j \in \text{NK}$ and $r_j \leq \hat{r}$, then $Q_j \in U_3$; if $S_j = \phi$, or $S_j \in \text{SK}$ and $r_j > \hat{r}$, or $S_j \in \text{NK}$ and $r_j > \hat{r}$, then $Q_j \in U_4$. The access rule of each operation type is described in Table 1.4.

In types 2 and 3, we only include queries with selectivity lower than the trigger level. Sequential processing (type 4) is preferred, if the number of tuples to be retrieved exceeds the trigger level, even though the desired tuples for a given query can be selected directly via the inverted directory (type 2) or can be projected through direct linkage between the two subfiles (type 3). We assume that the database designer determines the trigger level prior to the record segmentation decision, taking into account the trade-offs between random access and sequential access in terms of processing time.
Table 1.4. Internal-Level Classification of Operations

<table>
<thead>
<tr>
<th>Type</th>
<th>Access Rule</th>
<th>Conceptual-Level Query Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Find the page number of the tuple whose primary key value is specified in the selection formula by randomly searching the primary directory. Directly access this tuple in a subfile. Then sequentially access the remaining tuples in the response set of the query. (For a simple atomic query, this step is not required.) The subfile(s) to be accessed is determined according to the projection schema.</td>
<td>b.1.1, b.1.2</td>
</tr>
<tr>
<td>2</td>
<td>Find the TID's of tuples in the response set via the inverted directory. Then directly access the tuples using the given address list. The subfile(s) to be accessed is determined according to the projection schema.</td>
<td>b.2.1, b.2.2, b.2.3 - when ( r_j \leq r )</td>
</tr>
<tr>
<td>3</td>
<td>Sequentially scan the whole subfile which contains the selection attribute and find the TID's of tuples in the response set. Then directly access the other subfile, if it is required for projection.</td>
<td>b.3.1, b.3.2, b.3.3 - when ( r_j \leq r )</td>
</tr>
<tr>
<td>4</td>
<td>Sequentially access either or both subfiles. For conceptual-level categories a and b.2, the subfile(s) to be accessed is determined by the projection schema. For category b.3, the subfile containing the selection attribute has to be scanned. If access to the other subfile is necessary for projection, also scan that subfile.</td>
<td>a, b.2.1, b.2.2, b.2.3 - when ( r_j &gt; r ), b.3.1, b.3.2, b.3.3 - when ( r_j &gt; r )</td>
</tr>
</tbody>
</table>
I.5 Model Formulation

I.5.1. A generic framework

The objective of record segmentation is a reduction of computer operating time associated with database usage; that is, the cost model is measured in terms of operating time. Assuming that both subfiles are stored on the same type of device, we disregard the cost of secondary storage. We introduce an additional assumption to simplify the framework for the cost model.

Assumption 6 (Constant Block Accesses to Primary Directory): The primary directory is loaded on the external device after the record segmentation decision is made. While loading the primary key directory, the database designer determines the blocksize of index entries. This blocksize is selected in such a way that the number of block accesses to the primary directory for searching the desired page is restricted to some constant.

Theorem 1 Under Assumptions 2, 3 and 6, the record segmentation decision is indifferent to the cost of accessing index directories for random access. (See Appendix A for the proof.)
Corollary 1 A cost model formulated under Assumption 4 subsumes the special case of multiple key queries where selection attributes are all inverted for random access. (See Appendix B for the proof.)

A general model for optimizing the segmentation of a relation can be mathematically formalized in a canonical form as follows:

$$\min_{\mathbf{x}_s} Z = \sum_{t=1}^{T} \sum_{s=1}^{2} \sum_{j=1}^{\#U_t} f_j \ G_t (\mathbf{x}_s ; \alpha, \beta, \theta)$$  \hspace{1cm} (P')$$

A record segment can be labeled as segment 1 or segment 2 arbitrarily, because we have no predefined priority in accessing the subfiles. We will use the subscript $s$ ($s=1,2$) to indicate one of the two subfiles, $R_1$ and $R_2$. We define decision variables $x_{is}$, $i=1,2,\ldots,m$, with $m$ being the arity of the composite relation $R$, such that $x_{is} = 1$ if attribute $d_i$ is assigned to record segment $s$; $x_{is} = 0$ otherwise. A decision vector $\mathbf{x}_s$ is defined as an $m$-vector of $x_{is}$. (In this Part, we underline a symbol when it denotes a vector, and double-underline it when it denotes a matrix.)

In (P'), $T$ is the number of different operation types and $G_t$ is a functional expression of the cost of implementing the access rule (as in Table 1.4) for operation type $t$. In other words, $G_t$ is a function that relates $\mathbf{x}_s$, the state-space representation of a specific record segmentation, to the cost of processing each user request based on that segmentation. $\alpha$, $\beta$ and $\theta$ are sets of input parameters to the model. $\alpha$ is derived from the relation descriptor (Table 1.1) and given by $\alpha=(m, \ell_1$
(i=1,\ldots,m), N \). \( \beta \) is derived from the query descriptor (Table 1.3) and defined as \( \beta = (r_j, S_j, P_j) \). \( \Theta \) is a set of hardware parameters such as those shown in Table 1.5.

A transformation of the decision variables \( x_{is} \) is typically needed to construct new variables indicating the need for accessing subfile \( s \) to satisfy a query \( Q_j \) [12]. However, as the database structure becomes more complicated by augmenting access paths, and as the types of operations get diversified by enlarging the scope of user requests, we will need more transformations yielding new variables to indicate newly defined access rules. The generic model asserts that the functional expression of cost for processing user requests is the same for all requests that are classified into one operation type. It also asserts that the overall cost of database usage by the whole user community is simply the sum of the costs for individual operation types. Because of our paradigm of record segmentation, viz, no discrimination of one subfile from the other, the cost for each operation type is again a simple sum over two subfiles. Thus, the complexity of interactions among record segmentation, access path structuring and query optimization can be resolved in design Steps 2 and 3, where the operation types are carefully identified, and then the model formulation can be modularized - one cost function for each operation type.
1.5.2. The cost model

We assume that the database is stored in a movable head disk unit. In the total operating time, we include only the I/O time consisting of the seek time, latency time and the data transfer time. We exclude the head selection time, because it is negligible compared with the other I/O time factors. We also exclude the CPU time for the same reason. Storage device parameters for the cost model are listed in Table 1.5.

Table 1.5. Storage Device Parameters

| lp     | length in bytes of a page in the subfiles |
| cp     | number of pages in a cylinder             |
| Ta     | average seek time                         |
| Tl     | average latency time                       |
| Tm     | minimum seek time                         |
| Tt     | time required to transfer data in a page to main memory |

We present a formulation of function \( G_t \) for each of the four operation types, \( t=1,2,3,4 \), based on the access rules specified in Table 1.4.
I.5.2.1. Function $G_1$

Let $l_{R_s}$ be the length in bytes of a tuple in $R_s$, i.e., $l_{R_s} = \sum_{i=1}^{m} l_i x_{i_s}$. Given a decision vector $X_s$, the number of tuples $b_s$ in a page of $R_s$, i.e. the blocking factor of $R_s$, is calculated as $b_s = \lceil lp / l_{R_s} \rceil^{(4)}$, ignoring the inter-block gaps.

After locating the page address of a tuple whose primary key value is specified in the selection formula of the query, the access method retrieves the response set from the relevant subfile. With a simple atomic query, it directly retrieves one desired page. With a range atomic query, it first directly accesses the page whose address is given by the directory, and then sequentially retrieve all the pages that contain tuples in the response set. In any case, the number of pages to be accessed in a subfile $s$ will be $\lfloor r_j / b_s \rfloor$.

The subfile(s) to be accessed in order to satisfy a query is determined according to the projection schema. Define $y_{s_j} = 1$ if $R_s$ is required to be accessed for projection by a query $Q_j$; $y_{s_j} = 0$ otherwise. Also define $p_{i_j} = 1$ if $d_i \in P_j$, i.e., an attribute $d_i$ appears in the projection schema of a query $Q_j$; $p_{i_j} = 0$ otherwise. Then, $y_{s_j} = \min \{ 1, \bigwedge_{i \in P} p_{i_j} x_{i_s} \}$. Incorporating the hardware parameters, $G_1$ is formulated as follows:

$$G_1 = y_{s_j} \{ Ta + Tm \{ r_j / (b_s \ cp) \} + Tt \{ r_j / b_s \} \}$$
1.5.2.2. Function $G_2$

After locating the TID's of desired tuples for $Q_j$ via the inverted directory, $r_j$ tuples are accessed directly. However, since the tuples are blocked in units of pages, the expected number of page accesses to retrieve $r_j$ tuples from subfile $R_s$ is estimated, following Assumption 7 stated below, to be the total number of pages in $R_s$, or $NP_s = \lfloor N/b_s \rfloor$, multiplied by the probability that a page contains at least one out of $r_j$ tuples. Similarly, the expected number of cylinder accesses to retrieve $r_j$ tuples from $R_s$ is estimated to be the total number of cylinders assigned to $R_s$, or $NC_s = \lfloor N/(b_s cp) \rfloor$, multiplied by the probability that a cylinder contains at least one out of $r_j$ tuples.

Assumption 7 (Distribution of Tuples): Tuples are independently and uniformly distributed across a file with respect to the value of any attribute in the tuple.

The probability that a page (or a cylinder) contains at least one out of $r_j$ tuples is expressed as a function $H$ of the variables $r_j$ and $g_s$, where $g_s$ is the "grouping factor" or the number of tuples in one page (or one cylinder) of $R_s$. $H$ can be one of the following expressions according to the degree of accuracy required and the database characteristics.

$$H(g_s; r_j) = 1 - \sum_{i=1}^{r_j} \frac{(N-g_s-i+1)}{(N-i+1)} \quad \text{if } r_j \leq N-g_s$$

$$= 1 \quad \text{otherwise} \quad (1.1)$$
H(g_s; r_j) = 1 - (1 - N/M)^{r_j} + (1/M^2 g_s)(r_j(r_j-1)/2)(1 - (1 - N/M)^{r_j-1})\
\times (r_j(r_j-1)(2r_j-1)/6)(1 - (1 - N/M))^{r_j-1} \quad \text{if } r_j \leq N - g_s \quad (1.2)
\text{otherwise,}

where \( M = \lfloor N/g_s \rfloor \)

\[ H(g_s; r_j) = 1 - \left(1 - \frac{r_j}{N}\right)^{g_s} \quad (1.3) \]
\[ H(g_s; r_j) = 1 - \left(1 - \frac{g_s}{N}\right)^{r_j} \quad (1.4) \]

A detailed comparison of Eqs. (1.1) and (1.4) can be found in [137]. Eq. (1.2) is proposed in [133] and compared against all the other equations. In this Part, it suffices to point out that (i) Eq. (1.1) is an exact formula, but requires a relatively long computation time because of the iterative property of the formula, (ii) Eq. (1.2), a closed form approximation of (1.1), is computationally faster than (1.1), and provides negligible deviations (maximum error = 3.7%) from (1.1) over the entire range of \( g_s \) and \( r_j \), (iii) Eqs. (1.3) and (1.4) [19] are still faster than (1.2), but yield a good approximation only when \( g_s \ll N \) or \( r_j \gg 1 \) and when \( r_j \ll N \) or \( g_s \gg 1 \), respectively.

Therefore, the database designer should choose a formula for the \( H \) function that best fits his/her strategy as to the trade-off between computational accuracy and computation time, and is appropriate for the estimation in consideration. (6) Given an appropriate selection of formula for the \( H \) function, \( G_2 \) can be formulated as follows:

\[ G_2 = y_{sj} \left[ Ta + Tm \left\{ N C_s H(b_{scp}; r_j) - 1 \right\} + (Tt+Tt) N P_s H(b_s; r_j) \right] \]
The second term within the brackets is the time required for the read/write head to linearly move in one direction to the cylinders that contain at least one page to be retrieved. Since the address list of tuples to be retrieved is sorted in increasing order of the address (Assumption 3), the read/write head can move in one direction to the next cylinder containing pages with larger addresses, and hence we can apply the minimum seek time for each movement. Notice that we have ignored the time for the read/write head to bypass all cylinders between those containing the desired pages. The time to bypass a cylinder without retrieving any page is negligible relative to the time needed to access a cylinder for data retrieval. Also, for a reasonable size of selectivity, there is little chance of bypassing a cylinder anyway.

I.5.2.3. Function $G_j$

For a type 3 query, if the selection attribute is in one subfile and any attribute to be projected is in the other subfile, the former subfile is accessed sequentially, and then the latter subfile is accessed randomly for projection (Case 1). However, if all the attributes to be projected are in the same subfile as the selection attribute, only one subfile is to be accessed sequentially (Case 2).

$x_{S_jS_i}$, i.e. $x_{iS}$ for $i = S_j$, indicates the need for accessing a subfile $R_s$ for selecting the tuples in the response set of a query $Q_j$. That is, $x_{S_jS_i} = 1$ if the selection attribute of $Q_j$ is in $R_s$; $x_{S_jS_i} = 0$ otherwise. Define $z_{S_j} = 1$ if $Q_j$ needs to access $R_s$ for projection after
sequentially searching the other subfile; \( z_{sj} = 0 \) otherwise. Then, \( z_{sj} = 1 \)
only if \( y_{sj} = 1 \) and \( x_{sj} = 0 \) (Case 1), and \( z_{sj} = 0 \) otherwise (Case 2).

Using the sum-of-products expression, we can express \( z_{sj} \) as \( y_{sj} \left( 1 - x_{sj} \right) \). Now \( G_3 \) is formulated as follows:

\[
G_3 = x_{sj} \left\{ Ta + Tm \left( NC_s -1 \right) + Tt \left. NP_s \right\} + z_{sj} \left\{ Ta + Tm \times \left[ NC_s H(b_s; r_j) - 1 \right] + (Tt+Tt) \left. NP_s H(b_s; r_j) \right\} \right. 
\]

I.5.2.4. Function \( G_4 \)

The access method used for processing type 4 queries is sequential. However, as described in Table 1.4, the rule for determining the subfile(s) to be accessed is not identical for all the queries of this type. For queries with a secondary key selection and for queries without a selection formula, \( y_{sj} \) indicates the subfile to be accessed. For queries with a non-key selection, \( x_{sj} + z_{sj} \) indicates the subfile to be accessed. Incidentally, if \( Q_j \) has no selection formula, then \( x_{sj} = 0 \) for \( vs \) so that \( x_{sj} + z_{sj} = x_{sj} + \left( 1 - x_{sj} \right) y_{sj} - y_{sj} \). Therefore, a formulation of the cost function for non-key retrievals will hold for all the queries in type 4 if we also set \( x_{sj} = 0 \) for \( vs \) and \( Vj \) such that \( Q_j \) is a secondary key retrieval with selectivity greater than \( \hat{r} \). This can be done easily in any solution procedure. The resultant expression for \( G_4 \) is given as follows:

\[
G_4 = \left( x_{sj} + z_{sj} \right) \left\{ Ta + Tm \left( NC_s -1 \right) + Tt \left. NP_s \right\} \right. 
\]
I.5.2.5. Total cost function

The total cost function \( Z \) is obtained by substituting in \( (P') \) the four equations derived above for \( G_1 \) through \( G_4 \). The resulting expression for \( Z \) will be called Problem \( (P) \).

\[
\begin{align*}
\text{Min} & \quad Z = c \cdot V \\
& \quad \text{Subject to} \quad \sum_{s=1}^{2} \left[ \sum_{j \in U_1} c^1_{sj} y_{sj} + \sum_{j \in U_2} c^2_{sj} y_{sj} + \sum_{j \in U_3} (c^3_{sj} x_{sj} + c^4_{sj} z_{sj}) \\
& \quad + \sum_{j \in U_4} c^5_{sj} (x_{sj} + z_{sj}) \right] (1.6)
\end{align*}
\]

where

\[
\begin{align*}
c^1_{sj} &= f_j\left[ T_a + T_m(2C - 1) + T_t N_{P_s} \right], \quad \forall s, j \in U_1 (1.7) \\
c^2_{sj} &= f_j\left[ T_a + T_m \{ N_{C_s} H(b_s; r_j) - 1 \} \\
& \quad + (T_1+T_t) N_{P_s} H(b_s; r_j) \right], \quad \forall s, j \in U_2 (1.8) \\
c^3_{sj} &= f_j\left[ T_a + T_m(2C - 1) + T_t N_{P_s} \right], \quad \forall s, j \in U_3 (1.9) \\
c^4_{sj} &= f_j\left[ T_a + T_m \{ N_{C_s} H(b_s; r_j) - 1 \} \\
& \quad + (T_1+T_t) N_{P_s} H(b_s; r_j) \right], \quad \forall s, j \in U_3 (1.10) \\
c^5_{sj} &= f_j\left[ T_a + T_m(2C - 1) + T_t N_{P_s} \right], \quad \forall s, j \in U_4 (1.11) \\
y_{sj} &= \min \left\{ \frac{1}{m} \sum_{i=1}^{m} p_{ij} x_{is} \right\}, \quad \forall s, v_j (1.12) \\
z_{sj} &= y_{sj} \left(1 - x_{sj} \right), \quad \forall s, j \in \{U_3, U_4\} (1.13) \\
b_s &= \left[ \frac{1}{m} \sum_{i=1}^{m} x_{is} \right], \quad \forall s (1.14) \\
N_{P_s} &= \left[ N / b_s \right], \quad \forall s (1.15) \\
N_{C_s} &= \left[ N_{P_s} / c_p \right], \quad \forall s (1.16) \\
N_{P_s} &= \left[ r_j / b_s \right], \quad \forall s, j \in U_1 (1.17) \\
N_{C_s} &= \left[ N_{P_s} / c_p \right], \quad \forall s, j \in U_1 (1.18) \\
s=1,2, i=1,\ldots,m, j=1,\ldots,n
\end{align*}
\]
In the objective function (1.5), \( \mathbf{c} \) is a vector of functions \( c_{sj}^k \), \( k=1,\ldots,5 \), \( s=1,2 \), \( j=1,\ldots,n \). Note that \( c_{sj}^k \) is a linear function of \( \mathbf{lR}_s \) for \( k=1,3,5 \) (ignoring the ceiling and floor operators), whereas it is a complex nonlinear function of \( \mathbf{lR}_s \) for \( k=2,4 \) because of the term \( \mathbf{H} \).

\( \mathbf{lR}_s \) itself is a linear function of \( x_{is} \), \( i=1,2,\ldots,m \), namely, \( \mathbf{lR}_s = \sum_{i=1}^m \mathbf{l}_i x_{is} \). In the same equation, \( \mathbf{V} \) is a vector of all the transformed variables: \( y_{sj} \) for \( \mathbf{Vs} \) and \( j \in (U_1,U_2) \); \( x_{sj} \) and \( z_{sj} \) for \( \mathbf{Vs} \) and \( j \in (U_3,U_4) \).

Problem (P) can be rewritten as a 0-1 nonlinear program (NLP) by converting Eqs. (1.12) through (1.18) into functional constraints without non-algebraic operators such as the minimum operator, ceiling operator and the floor operator. We simply ignore the ceiling operator in Eq. (1.14). Problem (NLP) is formulated as follows:

\[
\begin{align*}
\text{Min} & \quad \{ \mathbf{W}, \mathbf{NP}_{s}, \mathbf{NC}_{s}, \mathbf{NP}'_{s}, \mathbf{NC}'_{s}, s=1,2 \} \\
\text{s.t.} & \quad \frac{2}{\mathbf{W}} \sum_{s=1}^{2} x_{is} = 1, \quad \forall i \\
& \quad y_{sj} - x_{is} \geq 0, \quad i \in P_j, \quad \forall s, \quad \forall j \\
& \quad z_{sj} - y_{sj} + x_{js} \geq 0, \quad \forall s, \quad j \in \{ U_3, U_4 \} \\
& \quad \mathbf{NP}_{s} - \mathbf{N} \sum_{i=1}^{m} \mathbf{l}_i x_{is} / \mathbf{l}_p \geq 0, \quad \forall s \\
& \quad \mathbf{NC}_{s} - \mathbf{NP}_{s} / \mathbf{c}_p \geq 0, \quad \forall s \\
& \quad \mathbf{NP}'_{sj} - \mathbf{r}_j \sum_{i=1}^{m} \mathbf{l}_i x_{is} / \mathbf{l}_p \geq 0, \quad \forall s, \quad j \in U_1 \\
& \quad \mathbf{NC}'_{sj} - \mathbf{NP}'_{sj} / \mathbf{c}_p \geq 0, \quad \forall s, \quad j \in U_1 \\
& \quad 0 \leq \mathbf{W} \leq 1, \mathbf{W} \text{ integer} \\
& \quad \mathbf{NP}_{s}, \mathbf{NC}_{s}, \text{ integer}, \quad \forall s \\
& \quad \mathbf{NP}'_{sj}, \mathbf{NC}'_{sj}, \text{ integer}, \quad \forall s, \quad j \in U_1 
\end{align*}
\]
In the objective function (1.19), $W$ is a decision vector defined as $[x_1, x_2, y_{sj} \text{ for } V_s \text{ and } j \in \{U_1, U_2\}, z_{sj} \text{ for } V_s \text{ and } j \in \{U_3, U_4\}]$. Note that the vector $W$ is different from the vector $V$, a vector of variables appearing in the objective function, in that $W$ includes all elements of $x_1$ and $x_2$, whereas $V$ includes only $x_{sjs}$, for $V_s$ and $j \epsilon \{U_3, U_4\}$, out of $x_1$ and $x_2$.

Problem (NLP) is equivalent to Problem (P) in that both problems have the same feasible set and produce the same optimal solution value, given the same parameter values. (It can be easily verified that the functional constraints in (NLP) and the corresponding equations in (P) define the same feasible set in the context of minimization.)
I.6 Solution Procedures

For problem instances with relatively few attributes (up to 20 or so), Problem (NLP) can be solved using exhaustive enumeration. However, because of the exponential growth of the search tree, exhaustive enumeration quickly becomes computationally intractable for larger problems.

In this chapter we present a heuristic procedure as well as an optimal solution algorithm that can be used to solve large problems. The heuristic procedure generates good feasible solutions with little computational effort. Furthermore, it can be used in conjunction with the optimal solution algorithm to significantly reduce the computational effort required for the latter. In spite of that, the optimal algorithm takes a relatively large amount of time; however, such an algorithm is needed to evaluate the performance of the heuristic. Also, in many instances the database designer may actually want to spend the extra computational time required to reach optimality. Especially with large databases, the amount of cost savings generated from the optimal segmentation may very well justify the additional time requirement of the optimal algorithm.
1.6.1. Heuristic solution procedure RSEGH

A local search approximation algorithm is developed as the heuristic, which we call RSEGH (Record Segmentation Heuristic). Local search algorithms are a simple and flexible method [48] for obtaining good solutions quickly. They have been successfully applied to NP-hard problems such as traveling salesman [82], general integer programming [52] and sequencing problems [8]. The reader is referred to [3; pp.336-343] for an excellent survey of local search algorithms.

The heuristic starts with a number of random segmentations and applies local transformations to each of the initial segmentations until reaching a locally optimal solution, one that no transformation can improve. These locally optimal solutions resulted from different starting segmentations are compared to choose the one that provides the lowest objective value. The following is a description of RSEGH in a Pascal-like pseudocode. RSEGH utilizes a sub-procedure named SUBAL.

**procedure RSEGH:**

```pascal
const m- ;  // the arity of a relation R
C- ;  // start value and increment value of the outer loop
NSD- ;  // iteration limit on the inner loop

type X=array[1..m] of binary integer;

var Xr, X2, X*, X*: x;
```

begin

\[ Z_h^* := \infty \quad \{ \text{initialize the heuristic value } Z_h^* \text{ at a large value} \} \]

for \( j := C \) to \( \lfloor m/2 \rfloor \) by \( C \) do \{ fix the number \( j \) of 1's in starting \( X_1 \) \}

for \( k := 1 \) to \( \text{NSD} \) do begin \{ generate a random starting solution \( X_1 \) that has \( j \) number of 1's in it \}

\( X_1[i] := 1, \text{ for } i = n_1, n_2, \ldots, n_j; \quad X_1[i] := 0, \text{ elsewhere}; \)

\( X_2 := \overline{X_1}; \quad \{ X_2 \text{ is a binary complement of } X_1 \} \)

\( \hat{Z}_h := Z(X_1, X_2); \quad \{ Z \text{ is a function that calculates the objective function value of problem (P) for the solution } (X_1, X_2) \} \)

SUBAL(\( X_1, X_2, \hat{Z}_h \)) \{ call subalgorithm SUBAL \}

if \( \hat{Z}_h < Z_h^* \) \{ if the objective value is improved by SUBAL \}

then begin \{ update the heuristic value and solution \}

\[ Z_h^* := \hat{Z}_h; \quad X_1^* := X_1; \quad X_2^* := X_2 \]

end

end; \{ RSEG \}
procedure SUBAL (var $X_1, X_2 : X; \var \tilde{Z}_h : \text{real}$);

begin

repeat

IMPROVED := false;

for $i := 1$ to $m$ do % for each attribute $i$ in the relation $R$

begin % check if the objective value is improved by reassignment

SWAP ($X_1[i], X_2[i]$); % SWAP is a procedure that exchange the values of two arguments

$Z_h := Z(X_1, X_2)$ % calculate the objective value of the changed solution

if $Z_h < \tilde{Z}_h$ % if the objective value is improved

then begin % update the incumbent value

$\tilde{Z}_h := Z_h;

\text{IMPROVED} := \text{true}$

end

else

SWAP ($X_1[i], X_2[i]$); % restore the assignment of attribute $i$ as before swapped

end

until not IMPROVED

end; % SUBAL
In the main procedure, first the number of attributes to be assigned to a subfile is fixed (outer loop). Then attributes are randomly selected to be included in the subfile (inner loop). It is found that the solution procedure works best when a number of different segment sizes (in terms of the number of attributes assigned to it) are used for starting segmentations.

Starting with the given initial segmentation, SUBAL checks, for every attribute, whether the objective value can be improved by a reassignment, i.e., by switching the attribute to the other segment. If the objective value is indeed improved by the switch, the algorithm fixes that reassignment; otherwise, it undoes that. In either case, the algorithm then continues with the next attribute. After all the attributes are considered in this way, if any one of them has been reassigned improving the objective value, then SUBAL iterates the same procedure of checking the attributes. SUBAL stops when the objective value cannot be improved by reassigning any attribute in one complete iteration.

The algorithm generates a total of \( \lceil \frac{m}{2} \rceil \cdot \text{NSD} \) initial segmentations. The worst case run time of SUBAL is \( m(m+1)/2 \). Therefore, RSEGH as a whole has a time complexity of \( O(m^3) \). The quality of the heuristic solution could improve at the expense of run time, as more starting solutions are tested for improvements. Computational results with RSEGH is reported in Chapter I.7.

A number of variations of local transformation method were implemented. One of these variations that has heuristic power comparable
to SUBAL is given in Appendix D. This subroutine named SUBAL1 is
interchangeable with SUBAL in RSEGH procedure. SUBAL1 reassigns a single
attribute that most improves the objective value among all the
attributes in each iteration, and backtracks to the earliest fixed
attribute once stuck in a local optimum. SUBAL1 exploits queue facility
to prevent indefinite backtracking. Because of backtracking SUBAL1 tends
to take more time than SUBAL, but it produced better solutions in some
problem instances. On the average, however, SUBAL slightly outperformed
SUBAL1 in our experiments.

1.6.2. Optimal solution algorithm

The complexity of Problem (NLP) is caused mainly by those terms
in the objective function that are nonlinear functions of segment
lengths. In this chapter we present an optimal solution procedure which
decomposes (NLP) into a series of easier 0-1 linear programs and derives
the optimal solution of (NLP) from the solutions of these subproblems.

1.6.2.1. Decomposition of (NLP)

The nonlinear program (NLP) can be decomposed into a series of
successive linear subproblems. Each subproblem (ILP) is a restriction of
(NLP) with an additional constraint that determines the lengths of
segments (constraint (1.31)). Let us denote a subproblem with constraint
\( l_{R1} = \ell \) as (ILP)\( _{\ell} \), which is formulated as follows:
where \( \ell \) is an integer in the set \( L^* = \{0, 1, 2, \ldots, \ell\} \), and \( 1(= \sum_{i=1}^{m} 1_i) \)

is the tuple length of the unsegmented relation \( R \). (ILP)_\ell becomes a problem of assigning attributes to a segment with fixed-length \( \ell \), and is a 0-1 linear program for which standard branch and bound solution techniques can be applied.

The restriction of (NLP) with constraint (1.31) leads to the following: (i) \( \mathbf{c} \), a vector of functions (1.7) through (1.11) of \( 1R_s \), becomes a constant vector in (ILP). Hence, the objective function (1.30) is converted from a very complex nonlinear function in (NLP) into a linear function of \( \mathbf{v} \) in (ILP). (ii) Terms (1.14) through (1.18) also become constants in (ILP), and hence constraints (1.23) through (1.26), as well as (1.28) and (1.29), become trivially satisfied. (iii) Such a restriction maps the huge search space \( \Omega = (X_1, X_2) \) with \( 2^{m-1} \) discrete states for (NLP) into a considerably reduced search space \( \Omega_\ell \) for each subproblem (ILP)_\ell such that \( \Omega_\ell = (X_1, X_2) \mid \sum_{i=1}^{m} 1_i x_{i1} = \ell, x_{i1} \in X_1, X_2 = X_1 \).

Since there are only a finite number of segment lengths, when all segment lengths are considered the best solution among optimal solutions of these subproblems will be the optimal solution for (NLP). Let

\[
\begin{align*}
Z^*(X_1, X_2) & = \min_{Z} \quad Z \quad \text{and} \\
Z^*_I(\ell) & = \min_{Z_I(\ell)} \quad Z_I(\ell) \quad \text{. Then, } \quad Z^*_I(\ell) \quad \text{is a function of } \ell \text{ on its integer domain } L^*, \quad \text{and } Z^* = \min_{\ell \in L^*} \quad Z^*_I(\ell) \quad \text{.}
\end{align*}
\]
Theorem 2 Function $Z^*_{I(\ell)}$ is symmetric about the point $\ell = 1/2$. (See Appendix C for the proof.)

Theorem 2 implies that it suffices to search only one half of $L^*$, i.e., the set of points to be searched could be $L = \{ 0, 1, ..., \lfloor 1/2 \rfloor \}$. Of course, it is possible that given a segment length, there is no feasible assignment of attributes to completely fill that segment. In that case we do not have to consider that subproblem any further.

Definition 3 (Feasible Tuple Length): A tuple length $\ell$ is said to be feasible, if there exists any $x_1$ such that $\sum_{i=1}^{n} l_i x_{i1} = \ell$ for $x_{i1} \in X_1$.

We can partition the set $L$ so that $L = \{ Lf, Li \}$, where $Lf = \{ \ell \ | \Omega_\ell \neq \emptyset \}$, a set of feasible segment lengths, and $Li = \{ \ell \ | \Omega_\ell = \emptyset \}$, a set of infeasible segment lengths. Consequently, by Theorem 2 and Definition 3, one could solve (ILP)$_\ell$, for $\forall \ell \in Lf$, in increasing order of $\ell$, to find the optimal solution to (P). Instances of (ILP)$_\ell$ for which $\ell \in Li$ are said to be "fathomed by infeasibility." A problem of testing the feasibility of a segment length is equivalent to a special case of the knapsack problem [56] where the utilities of individual items are proportional to their weights. Although such a knapsack problem is NP-complete [55], the average run time of the algorithm employed to solve it [3; p.67] was found to be less than a second for relations containing up to 40 attributes.
1.6.2.2. Strategies for fathoming (ILP) by bounds

Despite the fact that Problem (ILP) is easier to solve than Problem (NLP), the former is still NP-complete [55].(8) Furthermore, we have to solve $|L_f|$ of these subproblems. Therefore, some fathoming strategies are needed in the search process in order to expedite the solution procedure. A number of steps could be taken to speed up the process.

First, we pass the global incumbent value, $\hat{Z}$, or the objective value of the best known feasible solution to Problem (P), to successive subproblems as an upper bound on the optimal value of each subproblem. The initial incumbent solution is obtained by the heuristic RSECH, and the incumbent solution is updated whenever a better feasible solution to (P) is found.

Second, linear programming relaxations [56] and Lagrangian relaxations [50,51,60] are used to find lower bounds for each subproblem (ILP). These lower bounds are used in turn to quickly prune off subproblems, which cannot lead to better solutions, without searching for their optimalities. If one of these lower bounds is found to be larger than the current upper bound (the incumbent value), then any solution to that subproblem cannot be better than the existing solution and hence need not be considered any further. This is known as "fathoming by bound." Only when a subproblem is not fathomed, we solve it using a general branch-and-bound algorithm [56; ch.4], in which LP
relaxations again provide lower bounds and \( \hat{Z} \) is used as the initial upper bound.\(^9\)

A brief introduction to the relaxation techniques follows. LP relaxation of an integer program is obtained simply by dropping integrality constraints on decision variables. This technique is widely used when one wants to solve an integer program for optimality using a branch-and-bound algorithm [56]. Lagrangian relaxation is another scheme which can be successfully used to provide bounds in solving large-scale integer programs. Examples of the use of this technique include traveling salesman, scheduling, location, generalized assignment and set covering/partitioning problems (see [50] and its references).

Applications of Lagrangian relaxation to database systems research can be found in such areas as the design of distributed database systems [52,57] and query optimization [116]. Interested readers may refer to [50,51,60] for technical details on Lagrangian relaxation.

1.6.2.2.1. Linear programming relaxation of (ILP)

Let \((LP)_{\ell}\) denote Problem \((ILP)_{\ell}\) with the integrality constraints on \(W\) relaxed. Let us define \(Z^*_L(\ell)\) as the optimal objective value of \((LP)_{\ell}\). Then, \(Z^*_L(\ell)\) is a function of \(\ell\) on its integer domain \(L^*\). Theorem 2 holds also for \(Z^*_L(\ell)\). Typical plots of \(Z^*_L(\ell)\) and \(Z^*_L(\ell)\) on the domain \(L\) is shown in Figure 1.1. Except at a few segment lengths, \(Z^*_L(\ell)\) was observed to be almost unimodal on the domain \(L\) in all the problem instances. This nature of the function caused many of the subproblems to be fathomed by LP bounds. The portion of subproblems that can be
fathomed by LP bounds would increase, if the initial incumbent value $Z_h$ obtained by RSEGH is closer to the global optimal value, and if the gap between $Z^*_I(l)$ and $Z^*_L(l)$ is narrower.

In solving the (LP)'s, the optimal solution of the previously solved (LP) is converted to an initial basic solution for the next one. This causes the number of pivotings needed to solve an (LP) to decrease sharply. Let $\ell \in L_f$ be such that $(LP)_\ell$ is the most recently solved (LP). Suppose the current (LP) to solve is $(LP)_\lambda$, where $\lambda > \ell$. The change in problem structure leading from $(LP)_\ell$ to $(LP)_\lambda$ is limited to the RHS of constraint (1.31) and the objective coefficient vector $c$. Since this change is only slight, the optimal solution of $(LP)_\ell$ is likely to differ from that of $(LP)_\lambda$ only slightly. Actually, in cases when $\ell = \lambda + 1$, the initial basis for $(LP)_\ell$, generated from the optimal solution of $(LP)_\lambda$, was almost always found to be optimal for $(LP)_\lambda$ as well\(^{10}\). The reader is referred to [31; pp.160-161] for a discussion of this kind of procedure.

In the next subsection, we present a way to exploit the Lagrangian relaxation technique in order to further accelerate the process to search for the optimal solution.

I.6.2.2.2. Lagrangian relaxation of (ILP)

The following Lagrangian relaxation of $(ILP)_\ell$ is obtained by dualizing constraints (1.21) and (1.22), and dropping constraint (1.20). Constraint (1.20) is dropped by replacing $1 - x_{i1}$ for $x_{i2}$, for $Vi$. We will call this relaxation $(LR)_\ell$. 
Figure 1.1. Optimal Values, (LP) Bounds and (LR) Bounds of \((ILP)_x\) on the Domain \(L\)
s.t. (31)  
0 < W' < 1, W' integer,  
(1.33)
(LP) bounds. Consequently, we discarded this approach and adopted the following strategy: Whenever a subproblem is fathomed by an (LP) bound, we solve (LR)'s to get lower bounds for subsequent subproblems until a subproblem is not fathomed any further by the (LR) bound. We use the optimal dual values of the dualized constraints (1.21) and (1.22) in (LP) as the penalty vector \( \Delta \) in Lagrangian relaxations of subsequent subproblems, (LR)\(_{l+1}^\), (LR)\(_{l+2}^\), ..., (LR)\(_{l+k}^\), where in the \((l+k)\)th subproblem it is no longer possible to fathom by the Lagrangian bound. Such a deliberate choice of nonoptimal dual variables in subsequent subproblems is desirable in two respects: When \( \Delta \) is the optimal dual values in (LP)\(_l^\), (i) the gap between \( Z_L(l+1) \) and \( Z_D(l+1)(\Delta) \) increases only marginally as \( i=1,2,... \) increases, and (ii) the process for obtaining the lower bound \( Z_D(l+1)(\Delta) \) is very quick, since we solve only an instance of the knapsack problem with a given \( \Delta \). In Figure 1.1, typical drifts of \( Z_D(l+1) \), for \( i=1,2,...,k \) are shown by dotted lines.

I.6.2.3. Final algorithm RSEG

With the finer points explained above, we present here a formal statement of the optimal solution procedure RSEG. The algorithm checks every subproblem for infeasibility. Only feasible subproblems are solved successively according to the following rule: We start by solving the (LP) corresponding to the first feasible subproblem. If a subproblem (ILP) is fathomed by the (LP) lower bound, then we solve (LR)'s for the subsequent feasible subproblems until a subproblem is no longer fathomed by the (LR) lower bound. Once a subproblem is not fathomed by an (LR) lower bound, we solve the (LP) of the same subproblem, which is most
likely to produce a bound tighter than that (LR) bound. Whenever a subproblem is not fathomed by the (LP) lower bound, we solve the original (ILP) to obtain the optimal value of that subproblem and update, if possible, the upper bound for the following subproblems. If we have solved an (ILP), we again solve the (LP) of the next feasible subproblem for the optimality test. Even when a subproblem is not fathomed by the (LP) lower bound, the (LP) run time does not get wasted, because the (LP) solution is used as an initial basis for the branch-and-bound enumeration in solving the original (ILP).

**Algorithm RSEG**

**Input:** Relation descriptor, \(\{m, l_1, l_{i-1}, \ldots, l_m, N, PK, SK, NK\}\) from Step 2 of the design process

Query descriptor, query optimizer parameter and operation types, \(\{S_j, P_j, f_j, r_j, n\}\), \(r\), and \(U_t\) from Step 3 of the design process

Storage device descriptor, \(\{l_p, c_p, T_t, T_r, T_l, T_T\}\)

**Output:** \((X_1^*, X_2^*)\) and \(Z^*\)

**Method:** Successively solve the subproblems, with segment length varying from 0 to \(\lfloor 1/2 \rfloor\), using the fathoming strategies specified in previous sections.

**Step 1 (Initialization):**

Generate input parameters to Problem (P).

Solve (P) using Procedure RSEG to obtain a heuristic solution \((\hat{X}_1, \hat{X}_2)_h\) and its corresponding objective value \(Z_h\).

Set the initial incumbent value \(\hat{Z} = Z_h\) and the initial incumbent
solution \((X_1, X_2)^t = (X_1, X_2)^t_h\).

Set \(\ell = -1\).

**Step 2 (Construct the next feasible subproblem):**

Set \(\ell = \ell + 1\).

If \(\ell > \lfloor 1/2 \rfloor\), then goto Step 8.

Test the feasibility of \(\ell\) using the knapsack algorithm [3; p.67].

If \((\text{ILP})_{\ell}\) is fathomed by infeasibility, then start Step 2 again.

Otherwise, construct a new feasible subproblem \((\text{ILP})_{\ell}\).

Goto Step 3, if the previous step was Step 1 or Step 7.

Goto Step 5, if the previous step was Step 4 or Step 6.

**Step 3 (Construct and solve LP relaxation of a subproblem):**

Construct \((\text{LP})_{\ell}\) by dropping integrality constraints from \((\text{ILP})_{\ell}\).

If \(\ell = 0\), then solve the dual of \((\text{LP})_0\) from the scratch to obtain \(Z^*_L(0)\).

Otherwise, solve \((\text{LP})_{\ell}\) for \(Z^*_L(\ell)\) with the initial basis converted from the optimal solution to \((\text{LP})_{\ell}\).

Set \(\ell = \ell\).

Save the optimal solution \(\bar{w}^*_L(\ell)\) to \((\text{LP})_{\ell}\).

Save the vector of optimal dual values corresponding to constraints (1.21) and (1.22) in \((\text{LP})_{\ell}\), and label it as \(\Delta\).

**STEP 4 (Fathoming by LP lower bound):**

If \(Z^*_L(\ell) \geq \bar{z}\), then fathom \((\text{ILP})_{\ell}\) and goto Step 2.

Otherwise, goto Step 7.

**Step 5 (Construct and solve Lagrangian relaxation of a subproblem):**

Construct \((\text{LR})_{\ell}\) using \(\Delta\) that was saved in Step 3.

Solve \((\text{LR})_{\ell}\) for \(Z^*_D(\ell)(\Delta)\) using 0-1 knapsack algorithm [98].

**STEP 6 (Fathoming by Lagrangian lower bound):**
If $Z_D^*(\Delta_k) \geq \hat{Z}$, then fathom (ILP) and goto Step 2.
Otherwise, goto Step 3.

**STEP 7** (Solve the original subproblem and update the incumbent value):

Solve (ILP) for $(X_1^*,X_2^*)_{I(\ell)}$ and the optimal value $Z_I^*(\ell)$, using LP-based branch-and-bound algorithm [56] in which the initial cutoff value is set at $\hat{Z}$ and the LP optimal solution $\hat{W}_{L}(\hat{Z})$ saved in Step 3 is used as the initial basis. (Note that $\ell = \hat{\ell}$ in this step.)

If $Z_I^*(\ell) < \hat{Z}$, then set $\hat{Z} = Z_I^*(\ell)$ and $(X_1^*,X_2^*) = (X_1^*,X_2^*)_{I(\ell)}$. Goto Step 2.

**STEP 8** (Termination):

Terminate with $Z^* = \hat{Z}$, $(X_1^*,X_2^*) = (X_1^*,X_2^*)$. 

I.7 Computational Results

The heuristic procedure RSEGH was encoded in Fortran 77. Among the sub-procedures of Algorithm RSEG, the LP solution procedure and the branch-and-bound enumeration were implemented using Sciconic/VM, a state-of-the-art mathematical programming package, and the other sub-procedures were encoded in Fortran 77. Run on a Prime 9955 minicomputer, RSEGH and RSEG were applied to numerous record segmentation problems with varying parameters.

Table 1.6 provides a taxonomy of problem instances based on the experimental results reported. It was observed that the distribution of attribute lengths had a significant impact on the effectiveness of record segmentation as well as the expected run time of Algorithm RSEG. This is why the distribution of attribute lengths was chosen as a criterion to classify the problem instances. Problems with a high variance on attribute lengths are indicated as P1 and those with a low variance are indicated as P2. Other factors that significantly affect the effectiveness of record segmentation have been identified in the past research [47, 89, 92] and will be mentioned later in this chapter.
Table 1.6. Taxonomy of Problem Instances

<table>
<thead>
<tr>
<th>problem size (m)</th>
<th>distribution of attribute lengths</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>high variance (P1)</td>
</tr>
<tr>
<td>10</td>
<td>P1.10</td>
</tr>
<tr>
<td>20</td>
<td>P1.20</td>
</tr>
<tr>
<td>30</td>
<td>P1.30</td>
</tr>
<tr>
<td>40</td>
<td>P1.40</td>
</tr>
</tbody>
</table>

15 different problem instances were generated for each of the 8 problem categories listed in Table 1.6. The problem instances in a given category differed in the distribution of queries over 4 operation types, the bias towards the 80/20 rule, the selectivity of each query, the trigger level $r$ and the frequency of each query. The following parameters were fixed across the board: $n = 50$, $N = 100000$, $lp = 4096$ bytes, $cp = 57$, $Ta = .03$ sec, $Tm = .01$ sec, $Tl = .00834$ sec, $Tt = .00525$ sec (see Tables 1.1, 1.2 and 1.5). The attribute lengths in P1 (see Table 1.6) were randomly generated from a non-negative discrete distribution $\lfloor 50 \times |N(0,.5)| \rfloor + 1$, where $N(0,.5)$ is a normal distribution with mean 0 and standard deviation 0.5. The polar method for normal deviates [78] was employed to generate random numbers from $N(0,.5)$. The attribute lengths in P2 were generated in such a way that one tenth of the attributes had lengths that followed a discrete distribution $\lfloor 15 \times U(0,1) \rfloor + 1$, $U(0,1)$ being a uniform distribution.
over the interval \([0,1]\); for the remaining attributes, a discrete
distribution \([5 * U(0,1)] + 10\) was used.

Table 1.7 documents the gap between the objective value
(corresponding to the RSEGH solution and the true optimal value generated
by RSEG, as well as the expected run time of RSEGH. In every instance,
we used a value of 2 for \(C\), while \(m\) and NSD were as reported in the
table (see the description of RSEGH in Section 1.6.1). As the problem
size increased, we needed a larger number of starting solutions to
obtain a near-optimal solution.

Table 1.7. Analysis on the Gap and the Expected Run Time
of Heuristic Solutions

<table>
<thead>
<tr>
<th>problem size((m))</th>
<th>gap(%)*</th>
<th>expected run time(\text{sec.})</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSD</td>
<td>NS* min.</td>
<td>mean</td>
</tr>
<tr>
<td>10</td>
<td>5 10</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>10 50</td>
<td>0.00</td>
</tr>
<tr>
<td>30</td>
<td>20 140</td>
<td>0.00</td>
</tr>
<tr>
<td>40</td>
<td>40 400</td>
<td>0.00</td>
</tr>
</tbody>
</table>

* NS = total number of starting solutions tested

** gap = \{ (heuristic objective value - true optimal value) / true optimal value \} \times 100
RSEGH produced optimal solutions in all the problem instances up to a problem size of 20 attributes. For 30-attribute problems, the average gap of the heuristic value from the optimal value was negligible, and the maximum gap was 3%. For 40-attribute problems, the average gap was 1.4%, and the maximum gap was 5.4%. These gaps, of course, could be reduced further by increasing the number of starting solutions tested. Up to problems with 10 attributes, RSEGH was not significantly faster than exhaustive enumeration; beyond that point it became much faster than the latter. Input parameters other than the arity of the relation did not systematically affect the expected run time.

In Table 1.8, we summarize the average run time of Algorithm RSEG and some important factors that affected the run time. The tested problem instances are classified according to the taxonomy listed in Table 1.6. The reported run time is the total CPU time spent in running all the procedures of Algorithm RSEG.\textsuperscript{(13)} The run time was affected systematically by the following factors: the total number of subproblems, $|L| = \lceil L/2 \rceil + 1$; the portion of subproblems that were fathomed by the feasibility test, $|L_i|/|L|$; the portion of subproblems that were fathomed by the optimality test using an (LP) or an (LR) lower bound; and the portion of subproblems where the incumbent solution was updated.

The average run time per subproblem was 9, 17, 35 and 60 seconds for problems with 10, 20, 30 and 40 attributes in problem category P1. The corresponding figures for problems in category P2 were 11, 25, 41
and 68 seconds. It appears that the total run time of the algorithm is not less than that of exhaustive enumeration up to problems with about 15 attributes; beyond that point RSEG becomes considerably faster. The quality of the heuristic solution, which is used as the initial upper bound for RSEG, seems to be the most important contributor to the acceleration of the algorithm. This, together with the gap between $Z^*_{I(\ell)}$ and $Z^*_L(\ell)$, determines the portion of subproblems that are fathomed by lower bounds. It was observed that the gap between the two functions was wider when the distribution of attribute lengths was relatively uniform. This is the reason why the problem category P2 experienced systematically slower run times than P1, although P2 tended to have more infeasible subproblems than P1.

The cost saving realized by the optimal record segmentation is reported in Table 1.9. The cost saving is measured in percentage term as $100 \times (Z^0 - Z^*) / Z^*$, where $Z^*$ is the database usage cost under the optimal record segmentation and $Z^0$ is the same cost under no segmentation. We introduce another dimension into the taxonomy of problem instances -- a bias towards the 80/20 rule. The first row in the table represents a
<table>
<thead>
<tr>
<th></th>
<th>NSP</th>
<th>E (T)</th>
<th>E (T)/NSP</th>
<th>NIF/NSP</th>
<th>NFLP/NSP</th>
<th>NFLR/NSP</th>
<th>NNF/NSP</th>
<th>NUD/NSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(sec.)</td>
<td>(sec.)</td>
<td>(%)</td>
<td>(%)</td>
<td>(%)</td>
<td>(%)</td>
<td>(%)</td>
</tr>
<tr>
<td>min.</td>
<td>39.0</td>
<td>244.04</td>
<td>4.28</td>
<td>0.00</td>
<td>4.71</td>
<td>34.90</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>65.4</td>
<td>554.87</td>
<td>8.89</td>
<td>3.08</td>
<td>16.57</td>
<td>53.17</td>
<td>27.17</td>
<td>0.00</td>
</tr>
<tr>
<td>max.</td>
<td>106.0</td>
<td>839.21</td>
<td>15.35</td>
<td>9.43</td>
<td>26.47</td>
<td>78.94</td>
<td>56.75</td>
<td>0.00</td>
</tr>
<tr>
<td>std. dev.</td>
<td>19.6</td>
<td>172.89</td>
<td>3.33</td>
<td>3.22</td>
<td>9.02</td>
<td>16.88</td>
<td>24.04</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P1.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>57.0</td>
<td>545.16</td>
<td>9.39</td>
<td>18.96</td>
<td>3.50</td>
<td>7.01</td>
<td>46.55</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>57.5</td>
<td>639.25</td>
<td>11.12</td>
<td>23.51</td>
<td>6.07</td>
<td>16.91</td>
<td>53.49</td>
<td>0.00</td>
</tr>
<tr>
<td>max.</td>
<td>58.0</td>
<td>704.44</td>
<td>12.35</td>
<td>28.07</td>
<td>8.62</td>
<td>25.86</td>
<td>61.40</td>
<td>0.00</td>
</tr>
<tr>
<td>std. dev.</td>
<td>0.6</td>
<td>71.16</td>
<td>1.28</td>
<td>5.25</td>
<td>2.21</td>
<td>7.82</td>
<td>6.86</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>83.0</td>
<td>661.53</td>
<td>5.03</td>
<td>0.00</td>
<td>1.74</td>
<td>29.06</td>
<td>25.22</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>124.4</td>
<td>2097.12</td>
<td>17.02</td>
<td>0.33</td>
<td>6.06</td>
<td>44.12</td>
<td>49.50</td>
<td>0.00</td>
</tr>
<tr>
<td>max.</td>
<td>172.0</td>
<td>3990.12</td>
<td>29.51</td>
<td>1.16</td>
<td>12.61</td>
<td>62.16</td>
<td>68.02</td>
<td>0.00</td>
</tr>
<tr>
<td>std. dev.</td>
<td>31.8</td>
<td>1299.81</td>
<td>9.63</td>
<td>0.47</td>
<td>4.47</td>
<td>9.35</td>
<td>12.52</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P1.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>112.0</td>
<td>1735.65</td>
<td>15.49</td>
<td>8.03</td>
<td>1.68</td>
<td>22.32</td>
<td>58.82</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>115.5</td>
<td>2883.37</td>
<td>24.84</td>
<td>8.21</td>
<td>2.60</td>
<td>25.96</td>
<td>63.22</td>
<td>0.00</td>
</tr>
<tr>
<td>max.</td>
<td>119.0</td>
<td>3996.42</td>
<td>33.57</td>
<td>8.40</td>
<td>3.36</td>
<td>29.41</td>
<td>66.96</td>
<td>0.00</td>
</tr>
<tr>
<td>std. dev.</td>
<td>4.0</td>
<td>925.21</td>
<td>7.48</td>
<td>0.21</td>
<td>0.69</td>
<td>3.55</td>
<td>4.07</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>148.0</td>
<td>2826.38</td>
<td>17.23</td>
<td>0.00</td>
<td>1.45</td>
<td>24.36</td>
<td>24.50</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>194.2</td>
<td>6856.87</td>
<td>34.99</td>
<td>0.09</td>
<td>4.99</td>
<td>42.15</td>
<td>52.77</td>
<td>0.04</td>
</tr>
<tr>
<td>max.</td>
<td>275.0</td>
<td>12304.12</td>
<td>53.78</td>
<td>0.72</td>
<td>13.24</td>
<td>62.25</td>
<td>73.45</td>
<td>0.37</td>
</tr>
<tr>
<td>std. dev.</td>
<td>50.4</td>
<td>3030.63</td>
<td>12.21</td>
<td>0.25</td>
<td>4.00</td>
<td>11.48</td>
<td>14.97</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P1.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>166.0</td>
<td>4649.33</td>
<td>28.01</td>
<td>3.61</td>
<td>1.13</td>
<td>25.56</td>
<td>49.39</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>171.5</td>
<td>7104.25</td>
<td>41.28</td>
<td>4.64</td>
<td>3.55</td>
<td>31.87</td>
<td>59.94</td>
<td>0.00</td>
</tr>
<tr>
<td>max.</td>
<td>176.0</td>
<td>9775.09</td>
<td>55.53</td>
<td>5.68</td>
<td>6.62</td>
<td>40.36</td>
<td>67.61</td>
<td>0.00</td>
</tr>
<tr>
<td>std. dev.</td>
<td>5.8</td>
<td>2214.63</td>
<td>11.70</td>
<td>1.19</td>
<td>2.28</td>
<td>6.34</td>
<td>7.63</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>188.0</td>
<td>9743.18</td>
<td>44.90</td>
<td>0.00</td>
<td>2.09</td>
<td>23.40</td>
<td>46.08</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>239.2</td>
<td>13974.76</td>
<td>59.95</td>
<td>0.00</td>
<td>11.48</td>
<td>31.31</td>
<td>57.20</td>
<td>0.57</td>
</tr>
<tr>
<td>max.</td>
<td>335.0</td>
<td>17191.08</td>
<td>79.22</td>
<td>0.00</td>
<td>21.27</td>
<td>36.41</td>
<td>65.89</td>
<td>2.30</td>
</tr>
<tr>
<td>std. dev.</td>
<td>65.3</td>
<td>3484.78</td>
<td>15.92</td>
<td>0.00</td>
<td>10.48</td>
<td>5.66</td>
<td>8.58</td>
<td>1.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P1.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min.</td>
<td>220.0</td>
<td>10618.72</td>
<td>53.78</td>
<td>0.68</td>
<td>0.82</td>
<td>26.81</td>
<td>56.36</td>
<td>0.00</td>
</tr>
<tr>
<td>mean</td>
<td>227.5</td>
<td>16332.76</td>
<td>68.43</td>
<td>1.36</td>
<td>1.69</td>
<td>32.44</td>
<td>64.51</td>
<td>1.38</td>
</tr>
<tr>
<td>max.</td>
<td>235.0</td>
<td>23234.54</td>
<td>104.56</td>
<td>3.04</td>
<td>4.25</td>
<td>40.91</td>
<td>70.45</td>
<td>2.73</td>
</tr>
<tr>
<td>std. dev.</td>
<td>9.1</td>
<td>7922.85</td>
<td>34.54</td>
<td>0.53</td>
<td>0.81</td>
<td>9.96</td>
<td>9.75</td>
<td>1.46</td>
</tr>
</tbody>
</table>

E (T) = expected total run time of Algorithm RSEG
NSP = total number of subproblems, |L| = \left\lfloor L/2 \right\rfloor + 1
NIF = number of the subproblems fathomed by the feasibility test
NFLP = number of feasible subproblems fathomed by an (LP) lower bound
NFLR = number of feasible subproblems fathomed by an (LR) lower bound
NNF = number of feasible subproblems not fathomed
NUD = number of subproblems where the incumbent solution was updated
Table 1.9. Percentage Cost Savings by the Optimal Record Segmentation

<table>
<thead>
<tr>
<th>bias towards</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min.</td>
<td>mean</td>
</tr>
<tr>
<td>less biased</td>
<td>29.53</td>
<td>42.21</td>
</tr>
<tr>
<td>more biased</td>
<td>42.01</td>
<td>48.53</td>
</tr>
</tbody>
</table>

category of problem instances which are less biased towards the 80/20 rule than the instances in the second row. The database usage pattern for problem instances in the first category was such that 70% of all the attributes were not used as a selection attribute in more than 5% of all the queries, and 60% of them were not used as a projection attribute in more than 10% of the queries. The corresponding figures for problem instances in the second category were 80%, 5%, 70% and 10% respectively.

All 120 problem instances had both random processing and sequential processing queries, with neither being preponderant. The cost savings when both access methods were employed with similar weights was typically in the range of 30 - 60%. Record segmentation was found to be more effective when the database usage pattern was more akin to the 80/20 rule (this was also found in [92]), and when the distribution of attribute lengths had a higher variance. This latter observation implies that a database such as a statistical census whose attributes have relatively uniform lengths will receive less benefit from record
segmentation than a database such as a personnel database whose attributes have more variable lengths. A t-test on the mean of pairwise differences in the cost savings between problem categories P1 and P2 showed significance at the 95% confidence level. The same test performed between the problem categories with different biases towards the 80/20 rule also demonstrated a significant difference at the 99% confidence level. The arity of a logical relation was not observed to have a systematic impact on the effectiveness of record segmentation.

It was observed that the attribute assignments in the optimal record segmentation were typically such that the most actively used attributes were clustered together in one segment and extraordinarily long attributes tended to be assigned to the less active subfile. We also observed the following: Record segmentation was more effective, (i) the greater the ratio of the frequency of sequential processing queries to that of random processing queries was, (ii) as for randomly processed queries, the larger the selectivities were (also observed in [89]), and (iii) the larger the page length was.
In this Part we have presented an integrated model of record segmentation and access path selection. Several simplifying assumptions of the past research, e.g., a single access method and the dominance of one subfile over the other, are relaxed in this model. We have also discussed two solution strategies - a fast, near-optimal heuristic and a comprehensive optimal algorithm. Extensive computational results have been reported to show the effectiveness of these procedures.

The design process described in Chapter 1.3 and the generic framework for the cost model in Chapter 1.5 are developed so as to be applicable under more relaxed assumptions than are employed in this Part. Future studies can be directed towards extending our problem definition (Chapter 1.4) to include multiple key queries, updates, and binary operations such as join.

We believe our approach provides a better understanding of the underlying mechanisms of record segmentation in conjunction with other related issues in physical database design such as access path structuring and query optimization. It is hoped that the work presented in this Part will encourage and facilitate incorporation of record segmentation into a more realistic and unifying model of physical database.
PART II

A RULE-BASED QUERY OPTIMIZER

FOR VERTICALLY SEGMENTED RELATIONAL DATABASES
II.1 Introduction

Besides vertical partitioning of a database studied in Part I of this dissertation, database performance also critically depends on the access path structure which supports random access to desired tuples qualifying a query. As an example, many commercial and prototype DBMS's (e.g., ADABAS, IDMS/R, QBE, System R and System 2000) use indexed file structures. Direct access to desired blocks through certain levels of indirection is hence possible. Yet performance can be further enhanced by employing a query optimizer to determine the best access path for individual queries, as has been witnessed in INGRES [123] and System R [5, 117, 6, 23].

Query optimization therefore must be based on and fully utilize a given internal structure of the database. With record segmentation and random access paths both employed in a database, query optimization becomes considerably complex: selection of an access path given a query now should consider (i) whether one or both, and, if one, which subfile needs be accessed, (ii) if both are needed, which one should be first accessed, and (iii) for the subfile that must be first accessed, which access path is the minimal-cost path, (iv) after this subfile is processed, how to link the results to the second subfile, and finally
(v) how to process the second subfile. But none of the existing work has formally investigated query optimization in the context of vertically segmented databases.

The objective of this Part is to study optimization of one-variable queries (queries involving a single relation) on segmented relations. We assume the access path structure supports clustered and non-clustered indexes [5,117] and allows partial or full inversion of columns [111,63,4,70]. We employ basic query processing strategies such as the partial match retrieval using index list intersections [16,111,1,63,112], the direct linkage between subfiles [64,89,40], optimization of subfile access sequence [100,11,64] and the selection of an access method based on the size of a relational algebra [20,118,90,40]. As in System R, queries are considered to be in conjunctive normal form [119] and are optimized based on low-level (hardware dependent) analyses of the execution time.

We develop a query optimizer using the rule-based systems formalism [135,65]. Inputs to the rules are the parameters describing segmentation, indexing, and the query in consideration. The rules apply such factors as the sizes of relational algebras [63,117,42,108,27,54,87] and the rank [11] of each subfile in access sequence. These factors are computed using data in the system catalog. For a given query, the rules trigger a sequence of primitive operations which constitute an access module [23] for the query.
The model database system including the access path structure assumed in this Part is characterized in Chapter II.2, followed by the definitions of parameters describing a query in Chapter II.3. The rule-based query optimizer is developed in Chapter II.6 using these parameters and based on the query processing strategies discussed in Chapters II.4 and II.5. An example of query optimization using the rule-based system is given in Chapter II.7. Chapter II.8 provides some concluding remarks.
II.2 The Model DBMS and Access Path Structure

All three database models -- hierarchical, network and relational -- can support record segmentation [33,89,90]. Performance enhancement through efficient storage mapping and search strategies is especially important for the relational model when it is compared with other DBMS's [128]. Thus we adopt the relational model, particularly System R [5, 117,6,15,23], as the reference DBMS in this Part.

In System R, record segmentation is not automatically supported at the internal level. Rather, it is accomplished at the conceptual level with a relation split into two relations, each carrying the same primary key from the original relation. These two relations later must be explicitly specified in a query by the user, if they are needed.

In order to make record segmentation invisible to users at the external level, we require DBMS (i) maintain segmentation information (column assignments in each subfile) in the system catalog, (ii) ensure that tuples from the same record occurrence have the same relative addresses in respective subfiles [89,40], and (iii) employ a subfile linkage function (at the RSS level of System R, for example). A subfile linkage function computes the TID (Tuple IDentifier) of a tuple in one subfile directly from the TID of the corresponding tuple in the other.
subfile. Its benefits are: (i) redundant storage of the primary key is not necessary, (ii) search efficiency is enhanced (as will be discussed in Section II.6.2), and (iii) a single directory is sufficient for both subfiles.

Similar to images in System R, our index directories are $B^+$-trees [37], implemented as the VSAM Key-Sequenced Data Sets [73]. The order in which tuples are stored in a table is called the system sequence. This sequence is the sorted order of values of a column, if that column is chosen as the clustered index; it is the chronological order, otherwise. The system sequence is notable because if desired tuples can be retrieved in this sequence, then the sequential access method, due to its speed, should be used. Notice that the clustered index is a sparse index whereas non-clustered indexes are dense indexes. The latter is organized in the style of inverted directories [19]. The tuples are blocked in a fixed-length page of a virtual storage system on movable head disk units. Finally, the serial processing is assumed meaning that two subfiles cannot be processed simultaneously.
II.3 Query Parameters

Using the relational algebra [34], we express a query Q as

$$\pi_P \sigma_F (r),$$

where \( \pi \) is a projection operator, \( \sigma \) is a selection operator, \( P \)
is a projection schema (a set of projection attributes required to be
displayed), and \( F \) is a selection formula (e.g., a WHERE clause in SQL).

Any selection formula is considered to be in conjunctive normal form as
in System R Optimizer [117]. Let \( S \) be the set of all selection
attributes referenced in \( F \). Then we can define

$$F = \bigwedge_{i \in S} F_i.$$  

Every

conject \( F_i \) is called a Boolean factor and is defined such that

$$F_i = \bigvee_{j=1}^k \left( i \theta v[i]_j \right)$$

where \( v[i]_j \) is a value chosen from the domain of column

\( i \), and \( \theta \) is a relational operator \(<, =, >, \leq, \geq \) or \( \neq \).

Namely, \( F_i \) is a
disjunction of \( k \) atomic predicates relating the same column \( i \) to
different values. An index is said to "match" a Boolean factor if its
referenced column is indexed as a clustered or a non-clustered index
[117]. For simplicity, we assume \( k=1 \) for the Boolean factor matching the
clustered index.

Given an index selection, relation schema \( R \) can be partitioned
into subsets \( C, K \) and \( N \): \( C \) denotes a set of at most one column indexed
by the clustered index; \( K \), a set of columns indexed by non-clustered
indexes; and \( N \), a set of non-indexed columns. Set \( S \) can be partitioned
into subsets \( SC, SK \) and \( SN \) such that \( SC=S \cap C \), \( SK=S \cap K \) and \( SN=S \cap N \).
Given record segmentation, SN can be further partitioned into SN₁ and SN₂ such that SN₁ = SN₁ ∩ R₁ and SN₂ = SN₂ ∩ R₂. SN₁, for example, is a set of non-indexed columns in R₁ referenced by Q. Similarly, P can be partitioned into P₁ and P₂ such that P₁ = P₁ ∩ R₁ and P₂ = P₂ ∩ R₂.

We define a view to be a virtual table containing a set of all tuples in a subfile (or in the original relation) that qualify a relational algebra [109]. The size (cardinality) of a relational algebra or a view is the number of tuples that qualify the algebra or constitute the view [108]. A view is derived from the relation through a relational algebra which is a subexpression of the original query expression. This concept is important in query optimization developed in this Part: successive steps in query optimization basically reduce the size of views (intermediate results) towards the final response set of a query starting from the logical relation (see Figure 2.3 in Chapter II.5.). In the following two chapters, we will discuss the search strategies to be employed for query optimization.
II.4 Selection of an Access Method

It is well known in physical database design that, with a file stored on a moving head disk unit, there exists a trigger level of the selectivity beyond which a sequential scan of an entire search range is more efficient than the random retrieval of qualifying tuples using indexes [20,118,90,40]. The selectivity, $\beta/\alpha$ is the number $\beta$ of tuples to be retrieved from a search range in a table, divided by the size $\alpha$ of the range.

As an example, suppose the following: a table has 10,000 tuples; a query to be processed on this table has a selection formula "'SALARY > $35,000 and DEPT='Accounting' " where SALARY is a non-clustered index and DEPT is the clustered index; the number of tuples qualified for the condition "'DEPT='Accounting' " is 300 and that for the condition "'SALARY > $35,000 and DEPT='Accounting' " is 60. In this case, the system would retrieve the 60 tuples from the search range of size 300 if the random search is used. Hence, $\alpha=300$, $\beta=60$ and the selectivity of the query in the table is $\beta/\alpha=0.2$. Notice that if the sequential search is enforced, the system would need scan the search range of size 300 instead of 10,000 tuples in the entire table. The question is at which level of the selectivity the sequential search is less costly than the random search.
In this chapter we develop an exact solution for the trigger level -- it is the selectivity that equates the cost of sequentially scanning a search range (called the sequential search cost) to the cost of randomly retrieving the desired tuples from the range using indexes (called the random search cost). The trigger level is used in query optimization to determine the optimal access method for searching each subfile in processing a particular query.

In the following, we define the storage device parameters to be used in expressions for the random search cost and the sequential search cost. These costs are measured in I/O time, unless otherwise specified. Parameter values are computed assuming IBM 3330 disk unit. The reader is referred to [134] for formulas to compute the time parameter values.

\[ \ell_p = \text{length in bytes of a page} = 4096 \text{ bytes} \]
\[ cp = \text{number of pages in a cylinder} = 57 \]
\[ Ta = \text{average seek time} = .03 \text{ sec.} \]
\[ Tm = \text{minimum seek time} = .01 \text{ sec.} \]
\[ Ti = \text{average latency time} = .00834 \text{ sec.} \]
\[ Tt = \text{time required to transfer data in a page to main memory} \]
\[ = .00525 \text{ sec.} \]

The random search cost includes the index search cost and the table access cost. The index search cost to find a leaf index page containing a desired key value in the clustered-index directory is denoted as SCIND\_COST. Because the clustered index is sparse and structured as a B\(^{+}\)-tree, this search cost is
\[ SCIND\_COST = Ta + (Tl+Tt) \left( 1 + \log_{\ell_p} \left[ \frac{n}{B} \right] \right). \quad (2.1) \]

where \( n \) is the size of the table as defined earlier, \( \ell_a \) is the length in bytes of an index entry, i.e., a (key value, TID) pair, and \( B \) is the blocking factor of the table. In Eq. (2.1) we assume that the whole tree is contained in one cylinder.

The index search cost to obtain an intersected list of TIDs for the conjunction of all Boolean factors matching non-clustered indexes from the inverted directory is denoted as \( SIND\_COST \), which is

\[ SIND\_COST = Ta + (Tl+Tt)(1+\omega) + (Tl+Tt+Ti)\sum_{i=1}^{\omega} \mu_k, \quad (2.2) \]

where \( \omega = |SK| \), i.e., the number of non-clustered indexes matching \( Q \); \( \mu_k \) is the number of atomic predicates (i.e., the number of distinct key values) in the \( k \)th Boolean factor matching a non-clustered index; and \( Ti \) is the average time to intersect two pages of TID's. Note that \( Ti \) is measured in CPU time. Eq. (2.2) is based on the assumptions that the inverted directory has three levels (like the one illustrated in [19]) and is stored in one cylinder, and that a TID list for a (key, key value) pair is contained in one page. Readers can consult [19,49] for further details on the formulation of Eq. (2.2).

We now derive the table access cost based on the formula developed in [137] for computing the expected number of page accesses. Assume that the qualifying tuples are uniformly distributed among pages.
(or cylinders). Then the probability $H$ of a page (or a cylinder) containing at least one desired tuple is

$$H(\alpha, \beta; g) = 1, \text{ if } \beta > \alpha - g; \quad 1 - \binom{\alpha - g}{\beta}, \text{ otherwise.} \quad (2.3)$$

In Eq. (2.3), $g$ denotes the number of tuples in a page (i.e., blocking factor) or the number of tuples in a cylinder. Thus, we have $g = B$ for a page and $g = B \times cp$ for a cylinder.

The cost, $\text{RAND\_COST}(\alpha, \beta)$, of randomly retrieving $\beta$ tuples from a search range of size $\alpha$ in a table is accordingly formulated as

$$\text{RAND\_COST}(\alpha, \beta) = T_a + T_m \left( \left\lfloor \frac{\alpha}{(B \times cp)} \right\rfloor H(\alpha, \beta; B \times cp) - 1 \right) + (T_l + T_t) \left\lfloor \frac{\alpha}{B} \right\rfloor H(\alpha, \beta; B). \quad (2.4)$$

The expected number $\tilde{\alpha}$ of tuples that are retrieved when sequentially scanning a search range $\alpha$ for selecting $\beta$ tuples is derived in [101]:

$$\tilde{\alpha}(\alpha, \beta) = \beta (\alpha + 1) / (\beta + 1) \quad (2.5)$$

As $\beta$ increases, the difference between $\alpha$ and $\tilde{\alpha}$ becomes negligible, e.g., more than 90 percent of a search range has to be accessed when searching 9 or more tuples. Using Eq. (2.5), the cost of sequentially scanning a search range of size $\alpha$ for selecting $\beta$ tuples, denoted as $\text{SEQ\_COST}$, is

$$\text{SEQ\_COST}(\alpha, \beta) = T_a + T_m \left( \left\lfloor \frac{\tilde{\alpha}}{(B \times cp)} \right\rfloor - 1 \right) + T_t \left\lfloor \frac{\tilde{\alpha}}{B} \right\rfloor. \quad (2.6)$$
Now, let \( r \) denote the trigger level of the selectivity in processing \( Q \) in a table. Then, \( r \) is the solution to one of the following equations depending on whether \( Q \) matches the clustered-index (i.e., \( SC^\neq \phi \)) or not (i.e., \( SC=\phi \)). If \( SC^\neq \phi \), \( \alpha \) is the size of the relational algebra \( \sigma_{\bigwedge_i \in SC^I} \). If \( SC=\phi \), \( \alpha=n \).

\[
\text{SIND\_COST+RAND\_COST}(\alpha, \beta r) = \text{SEQ\_COST}(\alpha, \alpha r), \text{ if } SC=\phi. \tag{2.7}
\]

\[
\text{SCIND\_COST+SIND\_COST+RAND\_COST}(\alpha, \alpha r) = \text{SCIND\_COST+SEQ\_COST}(\alpha, \alpha r), \text{ if } SC^\neq \phi. \tag{2.8}
\]

Notice that \( SCIND\_COST \) is cancelled in both sides of Eq. (2.8) and hence does not affect the trigger level. As a consequence, Eq. (2.8) becomes identical to Eq. (2.7) except that \( \alpha \) assumes different values. Eq. (2.7) generally applies to a file of which the random search requires index search. In the context of segmented relations, it applies to the subfile which is first accessed in processing a query. In the sequel, we use notations \( r(I) \) and \( r(II) \) for the subfiles that are first and next accessed respectively. When one subfile is solely required for processing \( Q \), \( r(I) \) denotes that subfile.

The direct linkage between subfiles assumed in Chapter II.3 allows a random search of \( r(II) \) without revisiting the index directory. Thus, \( r \) for \( r(II) \) is the solution of

\[
\text{RAND\_COST}(\alpha, \alpha r) = \text{SEQ\_COST}(\alpha, \alpha r). \tag{2.9}
\]
Proposition 1 below states the optimal selection of an access method -- between sequential and random -- in processing Q in a table.

**Proposition 1** If $\beta/\alpha \geq r$ then the sequential access method should be used; otherwise, the random access method is preferred.

For each of Eq. (2.7) and (2.9), we performed 600 experiments for $\alpha=10, 10^2, 10^3, 10^4, 10^5, 10^6$, and for $B = 1, 2, \ldots, 100$, assuming $\omega_a=10, \omega=2, \mu_k=1$ for all $k=1, \ldots, \omega$, and $t_i=.02$ seconds. Experimental results for Eq. (2.7) and (2.9) are shown in Figures 2.1 and 2.2 respectively. Recall that Eq. (2.7) is applied to $r(I)$, and Eq. (2.9) to $r(II)$. We denote the blocking factor $B$ in $r(I)$ or $r(II)$ by $B(I)$ or $B(II)$.

In Figure 2.1, where index search is prerequisite for random processing, a sequential search is always preferred when a search range is small, say $\alpha < 100$. The reason is that the overhead of index search outweighs the benefit of random access even when $\beta$ is very small. When $\alpha > 1000$, on the other hand, the index search cost becomes orders of magnitude smaller than the table access cost, and hence Proposition 1 becomes effective for selecting a better access method.

In Figure 2.2, the absence of index search makes the random access more attractive: for example, when $\alpha=1000$, the random access can be chosen in the whole range 1-100 of $B(II)$, depending on the selectivity; for comparisons, if a search of the same size range were considered for $r(I)$, the sequential access would be preferred regardless of the selectivity as soon as $B(I)$ goes beyond 16 (see Figure 2.1).
It is apparent from both figures that \( r \) is inversely related with \( B \). On the other hand, \( r \) is almost indifferent to \( \alpha \) once the latter exceeds 1000. Comparing Figure 2.1 and 2.2, one can also see that when \( \alpha \) is large, the values of \( r \) in the two figures are almost the same over the entire range of \( B \). This again is because the index search cost is negligible relative to the table access cost when \( \alpha \) is large.

Based on the above, we can conjecture the following: (i) when \( \alpha < 100 \), the sequential access to a subfile, whether it is \( r(I) \) or \( r(II) \), is preferred regardless of \( \beta \) if \( B > 20 \); (ii) the trigger level \( r \) has to be estimated and Proposition 1 becomes effective if \( \alpha > 100 \) or \( B < 20 \); (iii) beyond a certain point of \( \alpha \), the values of \( r \) over the entire range of \( B \) become insensitive to the increase in \( \alpha \).

Given a segmented relation, blocking factors \( B(I) \) and \( B(II) \) in the respective subfiles are known. Now for a given blocking factor \( B(I) \) (resp. \( B(II) \)), we want to find the level of \( \alpha \), denoted as \( \alpha(I) \) (resp. \( \alpha(II) \)), below which the sequential access can be used unconditionally, and also the level of \( \alpha \), denoted as \( \tilde{\alpha}(I) \) (resp. \( \tilde{\alpha}(II) \)), beyond which the trigger level would remain the same as \( \alpha \) increases. In between \( \alpha \) and \( \tilde{\alpha} \), \( r \) may depend on the value of \( \alpha \), viz., on each query being processed.

For each of Eq. (2.7) and (2.9), we performed experiments for \( B(I) \) (or \( B(II) \))=10, 20, ... 100 to examine how \( r \) behaves as \( \alpha \) increases with fixed \( B(I) \) (or \( B(II) \)). \( \alpha \) was incremented by 10 between 10 and 100; by 100 between 200 and 1000; by 1000 between 2000 and 10,000. When \( \alpha \) is beyond 10,000, it was clear from the previous experiments that the
values of $r$ is insensitive to the increase in $\alpha$ and the index search cost. Experimental results for Eq. (2.7) and (2.9) are shown in Tables 2.1 and 2.2 respectively. In both tables, the values of $\alpha$ and $\tilde{\alpha}$ at each value of $B$ are marked by a solid and a broken line respectively. The line for $\alpha$ is drawn so that all $r$'s above it are such that $\alpha r < 2$, i.e., the trigger level in terms of the number of tuples is less than two. The line for $\tilde{\alpha}$, on the other hand, is drawn so that the gap between the maximum $r$ and the minimum $r$ below the line is at most 0.3%.

These tables confirm our conjectures and provide additional information: (i) $\alpha(I)$ and $\tilde{\alpha}(I)$ are each about an order of magnitude larger than $\alpha(II)$ and $\tilde{\alpha}(II)$; (ii) both $\alpha(I)-\alpha(I)$ and $\tilde{\alpha}(II)-\alpha(II)$ converge to zero as $B(I)$ and $B(II)$ increase; and (iii) the differences in the values of $r$'s in between $\tilde{\alpha}$ and $\alpha$ in both tables are negligible (at most 0.4%) except for
Figure 2.2. The Trigger Level of the Selectivity: Eq. (2.9)
Figure 2.1. The Trigger Level of the Selectivity: Eq. (2.7)
Table 2.1. The Trigger Level of the Selectivity: Eq. (2.7)

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>20</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>30</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>40</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>50</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>60</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
</tr>
<tr>
<td>70</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>80</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>90</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>200</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>300</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>400</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>500</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>600</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>700</td>
<td>2.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>800</td>
<td>2.8</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>900</td>
<td>2.9</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>1000</td>
<td>3.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>2000</td>
<td>4.0</td>
<td>1.6</td>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>3000</td>
<td>4.3</td>
<td>1.9</td>
<td>1.1</td>
<td>0.7</td>
<td>0.4</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>4000</td>
<td>4.4</td>
<td>2.0</td>
<td>1.2</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>5000</td>
<td>4.5</td>
<td>2.1</td>
<td>1.3</td>
<td>0.9</td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>6000</td>
<td>4.5</td>
<td>2.2</td>
<td>1.4</td>
<td>1.0</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>7000</td>
<td>4.6</td>
<td>2.2</td>
<td>1.4</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>8000</td>
<td>4.6</td>
<td>2.2</td>
<td>1.4</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>9000</td>
<td>4.6</td>
<td>2.3</td>
<td>1.5</td>
<td>1.1</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>10000</td>
<td>4.6</td>
<td>2.3</td>
<td>1.5</td>
<td>1.1</td>
<td>0.9</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Note: $r$ in the table is expressed as percentage.

Legend: \( \alpha(I) \) \hspace{1cm} \( \bar{\alpha}(I) \)
Table 2.2. The Trigger Level of the Selectivity: Eq. (2.9)

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>20</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>30</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>40</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>50</td>
<td>3.1</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>60</td>
<td>2.6</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
<td>1.7</td>
</tr>
<tr>
<td>70</td>
<td>3.6</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>80</td>
<td>3.2</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>90</td>
<td>2.8</td>
<td>1.7</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>100</td>
<td>3.6</td>
<td>1.6</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>200</td>
<td>4.3</td>
<td>1.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>300</td>
<td>4.2</td>
<td>1.8</td>
<td>1.2</td>
<td>0.5</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>400</td>
<td>4.4</td>
<td>2.1</td>
<td>1.4</td>
<td>0.9</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>500</td>
<td>4.5</td>
<td>2.1</td>
<td>1.3</td>
<td>0.9</td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>600</td>
<td>4.6</td>
<td>2.3</td>
<td>1.4</td>
<td>0.9</td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>700</td>
<td>4.5</td>
<td>2.2</td>
<td>1.4</td>
<td>1.1</td>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>800</td>
<td>4.6</td>
<td>2.2</td>
<td>1.4</td>
<td>1.1</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>900</td>
<td>4.6</td>
<td>2.3</td>
<td>1.5</td>
<td>1.1</td>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>1000</td>
<td>4.7</td>
<td>2.3</td>
<td>1.5</td>
<td>1.1</td>
<td>0.9</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>2000</td>
<td>4.7</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>3000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>4000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>5000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>6000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>7000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>8000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>9000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>10000</td>
<td>4.8</td>
<td>2.4</td>
<td>1.6</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Note: \( r \) in the table is expressed as percentage.

Legend: \( \alpha(II) \quad \bar{\alpha}(II) \)
B ≤ 10. Most likely the blocking factor is greater than 10 even for an unsegmented relation. Therefore, we can reasonably assume that r is independent of α, when α > q, i.e., in the whole range of α where Proposition 1 applies effectively. As a consequence, we have the following proposition:

Proposition 2 If α < q, then r=0. Otherwise, r is a constant for each subfile, determined by the blocking factor.

The database administrator can determine this constant value for each subfile by solving Eq. (2.7) or (2.9) with appropriate parameter values. We assume this constant value of r and the value of α for each subfile are maintained in the system catalog. Given a query, for each subfile the optimizer either sets r to be zero or retrieves its value from the catalog according to Proposition 2. (Later, when necessary, the trigger level of the selectivity will be denoted as τ₁ and τ₂ for subfiles r₁ and r₂, respectively.) Then Proposition 1 is applied for selecting the optimal access method. The values of β in the respective subfiles, which are required in Proposition 1, depend on the access path employed for searching them. In the next chapter, we explain the strategies of selecting access paths for r(I) and r(II), and also elaborate on the parameter β.
II.5 Selection of Access Paths

The strategies for selecting access paths in sequential and random searches depend on how the system utilizes indexes in processing queries. Also, because of the direct linkage between subfiles assumed in Chapter II.2, the search strategy is substantially different between \( r(I) \) and \( r(II) \). Processing \( r(I) \) is basically the same as searching an unsegmented relation.

In the sequel, we define \( SN(I) = SN \cap R(I) \) and \( SN(II) = SN \cap R(II) \). We also define \( \beta(I) \) and \( \beta(II) \) to be the actual number of tuples to be retrieved from subfiles \( r(I) \) and \( r(II) \). The determination of \( r(I) \) and \( r(II) \), viz., the optimal access sequence, depends on the search strategies presented in this chapter and will be discussed later in Chapter II.6.

II.5.1. Search strategies for processing \( r(I) \)

We now describe the strategy of using indexes in searching \( r(I) \). First, if \( Q \) matches the clustered index, i.e., if \( SC \neq \emptyset \), the system unconditionally accesses the clustered index directory and find the range of TID's qualifying the Boolean factor matching the clustered index. This operation, called \( SCIND \), is described below:
SCIND Anchor point(s) that point to the first and/or the last tuple of the search range defined by the Boolean factor matching the clustered index is obtained from the clustered index directory.

As mentioned earlier, the search range in both subfiles are reduced by SCIND so that $\alpha$ becomes the size of view $\sigma_{A_{1}\epsilon SC_{1}}(^{F_{1}}(r))$ (see Step 2 in Figure 2.3). When the primary key is selected as the clustered index and an equality atomic predicate is defined on this key in Q, the search range collapses to a single tuple and this tuple should be retrieved directly. The cost of implementing SCIND is formulated as SCIND_COST in the previous chapter.

Unlike the query optimizer in System R, if the random access is chosen for searching $r(I)$ according to Proposition 1, all non-clustered indexes matching Q are utilized to construct the smallest view (actually a list of TID's) that can be obtained through the inverted directory without accessing subfiles themselves. This search strategy, often called the partial match retrieval [128], can save orders of magnitude in query processing time under well selected indexes and has been adopted in [16,1]. Even if we assume, for compatibility with System R, that the system chooses only one index to utilize in processing Q, the query optimizer developed in this Part is still applicable. In this case, the optimizer needs incorporate an algorithm for selecting a single index for each query, such as the one used by System R Optimizer [5,128]. Accordingly, SC, SK and SN have to be redefined; let $k$ denote
the index chosen for Q; if \( k \in C \), then \( SC = \{ k \} \), \( SK = \phi \) and \( SN = S - SC \); if \( k \in K \), then \( SC = \phi \), \( SK = \{ k \} \) and \( SN = S - SK \).

The operations required for partial match retrieval is described below and their cost is formulated as SIND_COt in the previous chapter.

**SIND** For every non-clustered index matching Q, a list of TID's qualifying the matching Boolean factor is formed by searching the inverted directory.

**INT** These lists are intersected into a sorted list. If SCIND has been performed, the latter list is further intersected with the anchor points.

Note that operation INT is invoked only when \( |SK| \geq 2 \), or \( SC \neq \phi \) and \( |SK| \geq 1 \). A set of tuples corresponding to the final sorted list are retrieved from \( r(I) \). During these retrievals, false drops (retrieved tuples subsequently discovered not to belong to the response set of Q [77,128]) are filtered out by applying the Boolean factors referencing the columns in \( SN(I) \). Therefore, the tuples to be retrieved from \( r(I) \) corresponds to the view \( \sigma_{A_{1} \in SCUSK I}^{F_{1}}(r(I)) \), while the view obtained after processing \( r(I) \), called view i in the sequel, is \( \sigma_{A_{1} \in SCUSKUSN(I)}^{F_{1}}(r(I)) \) (see Step 3 in Figure 2.3). It is now clear that \( \beta(I) \) is the size of \( \sigma_{A_{1} \in SCUSK I}^{F_{1}}(r(I)) \).
II.5.2. Search strategies for processing r(II)

When r(II) is accessed after processing r(I), the list of TID's constituting view i is one-to-one mapped through the subfile linkage function to a corresponding list of TID's in r(II). If random access is chosen for searching r(II) according to Proposition 1, the tuples pointed by the latter list can be directly retrieved from r(II). \( \beta(II) \) therefore is the size of view i. Further qualification of the retrieved tuples by the Boolean factors referencing the columns in SN(II) will lead to the final response set of Q, which is represented as view ii in the sequel (see Step 4 in Figure 2.3). On the other hand, when a search range is defined by SCIND and a sequential scan of this range is enforced, the system needs the TID's in r(II) only for the anchor points. Thus, we can identify two different ways of subfile linkage:

\begin{itemize}
  \item **LINKF** If random access is chosen to search r(II), the TID's constituting view i is mapped to corresponding TID's in r(II).
  \item **LINKP** If Q matches the clustered index and the sequential access is chosen to search r(II), only TID's of the anchor points are mapped.
\end{itemize}

Notice that if Q does not match the clustered index and a sequential scan of r(II) is enforced, the subfile linkage is not necessary. Upon completion of retrieving tuples from r(II), if further selection has been made in r(II) and any projection attribute exists in r(I), the TID's of view ii should in turn be intersected with that of view i (see
Step 5 in Figure 2.3). We assume that the values of projection attributes selected in $r(I)$ are kept in main memory while $r(II)$ is searched. By doing so, additional I/O time required for projecting back to $r(I)$ can be saved. In the query optimizer shown in Table 2.3, this final step has been omitted because it does not require any database access.

Figure 2.3 demonstrates successive steps in query optimization in terms of views. In Step 4 and 5 of the figure, function $f$ in relational algebras is defined to be the one-one correspondence between TID's in the respective subfiles. Thus, $f(i)$ (resp. $f(ii)$) in Step 4 (resp. Step 5) maps the TID's of view i (resp. view ii) to corresponding TID's in $r(II)$ (resp. $r(I)$). The rules of query optimization, shown in Table 2.3, corresponding to each step are also referenced in the figure.

In order to apply Proposition 1 to select an access method before determining the actual access path to be used in searching each subfile, the system needs to estimate the values of $\alpha$, $\beta(I)$ and $\beta(II)$ for each query. We assume that the sizes of relational algebras that are required in query optimization can be estimated a priori with reasonable accuracy. Discussions of the method for estimating the size of a relational algebra is omitted here and the reader is referred to [108,27,54,28,87].
Figure 2.3. The Correspondence between Views and Rules in Query Optimization
II.6 Query Optimization

As mentioned in Chapter II.1, when record segmentation and random access paths are both employed in a database, a query optimizer should determine:

D.1 which subfile(s) to access in processing a query Q;
D.2 in which order to access two subfiles, if both are required to answer Q;
D.3 which access path to use for processing r(I);
D.4 which access path to use for processing r(II).

In the following, we elaborate on these decisions. The particular framework we choose to model knowledge for query optimization decisions is the so called rule-based approach in artificial intelligence. This approach has several useful features: (i) it provides a modular, but rigorous, formalism for knowledge organization, (ii) the execution of rules readily renders a plan for realizing a goal, and (iii) it is well supported, in a sense that conversion from rules to a program does not require special programming efforts. In the query optimization context, the goal is to minimize query processing time and a plan consists of a sequence of database operations. An example of a plan for a certain query is ‘‘search clustered index; search non-clustered index;"
intersect TID lists; randomly search r2; link TID's of the anchor points in r2 to those in r1; sequentially search the range of r1 defined by the anchor point'' (see Example 2 in Chapter II.7). In Appendix E, we show an implementation of the query optimization rules in LISP.

Syntactically, a rule is an expression of LHS \rightarrow [<OP>] RHS, where LHS is an antecedent or precondition, RHS a consequent or a post-condition and <OP> represents one or more operators. Operators are optional in a rule: a rule is called a production rule if it has operators; otherwise, it is an implication rule. In a rule-based system, a rule becomes an instruction and is executed when its antecedent becomes true. The effect of executing a rule is that RHS is inferred if the rule is an implication rule, or created (by the operators) if the rule is a production rule.

Table 2.3 summarizes the rules we need in query optimization. Rules 1 to 9 are implication rules which, except Rule 9, determine the optimal subfile access sequence, i.e., decisions (D.1) and (D.2). The rest of the rules are production rules. The operators in the production rules are the database operations which could constitute an access module for a query. The postcondition of a production rule plays a dual role in query optimization: first, it is a condition which, when becomes true, may subsequently invoke other rules; and second, it represents the result of the operators executed. For example, CIND of Rule 10 connotes (i) a condition that becomes true when SCIND is invoked, and (ii) a set of TID's returned by SCIND, i.e., the view $\sigma_{A_i \in SC_i}^F(x)$. The second role
is consistent with the concept of view we discussed (see Figure 2.3 for the correspondence between views and the rules). In the following section, we elaborate on decisions (D.1) and (D.2).

II.6.1. Optimal subfile access sequence

According to the search strategies presented in Chapter II.5, the TID's of tuples which satisfy the Boolean factors matching indexes can be determined by searching the index directory alone. The subfile search is needed, however, for selecting tuples based on the Boolean factors referencing non-indexed columns. Thus, a subfile needs be accessed in processing Q only if it contains at least a projection attribute or a non-indexed column referenced by Q, as summarized in the following proposition:

**Proposition 3** $r_1$ (resp. $r_2$) needs be accessed in processing $Q$, iff $SN_1 \cap P_1 \neq \emptyset$ (resp. $SN_2 \cap P_2 \neq \emptyset$).

Proposition 3 corresponds to Step 1 in Figure 2.3 and is represented as Rules 1, 2, and 3 in Table 2.3. In the table, $I_1$ and $I_2$ stand for conditions $r(I) = r_1$ and $r(I) = r_2$. $I_1$ and $I_2$, on the other hand, represent conditions $r(II) = r_1$ and $r(II) = r_2$. BOTH, which is inferred if $SN_1 P_1 \neq \emptyset$ and $SN_2 P_2 \neq \emptyset$, indicates that both subfiles, $r_1$ and $r_2$, need be accessed. When both subfiles need be accessed for processing $Q$, the order of access could be important.
Table 2.3. Rules for Query Optimization

<table>
<thead>
<tr>
<th>Rules</th>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Preconditions</td>
<td>Operators</td>
</tr>
<tr>
<td>1</td>
<td>(SN1 U P1 ≠ ∅), (SN2 U P2 = ∅)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(SN2 U P2 ≠ ∅), (SN1 U P1 = ∅)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(SN1 U P1 ≠ ∅), (SN2 U P2 ≠ ∅)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(P1 ≠ ∅), (P2 ≠ ∅), (SN = 0)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>BOTH, (SN1 ≠ ∅), (SN2 = 0)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>BOTH, (SN2 ≠ ∅), (SN1 = 0)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BOTH, (SN1 ≠ ∅), (SN2 ≠ ∅), (ρ(r1) ≤ ρ(r2))</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BOTH, (SN1 ≠ ∅), (SN2 ≠ ∅), (ρ(r1) &gt; ρ(r2))</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>(SH = ∅), (SN = 0)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>(SC ≠ ∅)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>I1, CIND, (SK = 0)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>I1, ~ CIND, (SK = 0)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>I1, CIND, (SK ≠ ∅), (γk/γ ≥ τ1)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>I1, ~ CIND, (SK ≠ ∅), (γk/n ≥ τ1)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>I1, CIND, (SK ≠ ∅), (γk/γ &lt; τ1)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>I1, ~ CIND, (SK ≠ ∅), (γk/n &lt; τ1)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>i, I2, NRAND, (SC = ∅)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>i, I2, NRAND, (SC ≠ ∅)</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>i, I2, ~ NRAND, (SC ≠ ∅), (γ1/n ≤ τ2)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>i, I2, ~ NRAND, (SC ≠ ∅), (γ1/n &lt; τ2)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>i, I2, ~ NRAND, (SC ≠ ∅), (γ1/n ≥ τ2)</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>i, I2, ~ NRAND, (SC = ∅), (γ1/n &lt; τ2)</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>I2, CIND, (SK = 0)</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>I2, ~ CIND, (SK = 0)</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>I2, CIND, (SK ≠ ∅), (γk/γ ≥ τ2)</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>I2, ~ CIND, (SK ≠ ∅), (γk/n ≥ τ2)</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>I2, CIND, (SK ≠ ∅), (γk/γ &lt; τ1)</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>I2, ~ CIND, (SK ≠ ∅), (γk/n &lt; τ2)</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>i, I1, NRAND, (SC = ∅)</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>i, I1, NRAND, (SC ≠ ∅)</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>i, I1, ~ NRAND, (SC ≠ ∅), (γk/n ≤ τ1)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>i, I1, ~ NRAND, (SC = ∅), (γk/n &lt; τ1)</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>i, I1, ~ NRAND, (SC ≠ ∅), (γk/n ≥ τ1)</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>i, I1, ~ NRAND, (SC = ∅), (γk/n ≥ τ1)</td>
<td></td>
</tr>
</tbody>
</table>
Lemma 1  When both subfiles need be accessed, the access sequence does not matter if $P_1 \neq \phi$, $P_2 \neq \phi$ and $SN = \phi$.

Proof  It is easy to see that either sequence results in the same number of page accesses. □

The corresponding rule is Rule 4. Notice that in that rule we arbitrarily assigned $r_1$ to be $r(I)$.

In the sequel, the notations for the sizes of relational algebras summarized in Table 2.4 are used. The size of each relational algebra in the table is defined to be $n$ if the algebra produces an empty set of qualifying tuples for a particular query. For example, if $Q$ matches no indexes, $\gamma^c \cdots \gamma^k \cdots \gamma^{ck} = n$.

Lemma 2  When both subfiles need be accessed, if $SN_1 \neq \phi$ and $SN_2 = \phi$ (resp. $SN_2 \neq \phi$ and $SN_1 = \phi$) then first accessing $r_1$ (resp. $r_2$) is optimal. (Rules 5 and 6)

Proof  The total number of tuples retrieved is $\gamma^{ck} + \gamma$ if the specified access sequence is followed; $2\gamma^{ck}$ if the alternative sequence is followed. Because $\gamma^{ck} > \gamma$, the former sequence is more efficient. In this case, the blocking factor in each subfile does not affect the optimal sequence, because the difference in tuple accesses between the two sequences occurs only in the subfile containing the non-indexed columns. □
Lemma 3 Let $B_1$ (resp. $B_2$) denote the blocking factor of $r_1$ (resp. $r_2$). Define the 'rank' $\rho$ of $r_1$ and $r_2$ such that $\rho(r_1) = \min\{n/B_1, c_1^k\} + \gamma_1$ and $\rho(r_2) = \min\{n/B_2, c_2^k\} + \gamma_2$, respectively. If $SN_1 \neq \phi$ and $SN_2 \neq \phi$, then the access sequence where a subfile with a lower rank is accessed first minimizes the worst case number of page accesses in processing $Q$. (Rules 7 and 8)

Proof The worst case number of page accesses is the minimum between the total number of pages in a subfile and the number of tuples to be retrieved from the subfile. It is proved in [11] that if the rank of a subfile is defined to be the worst case number $k_1$ of page accesses in the subfile plus the number $k_2$ of tuples finally selected from the same subfile, then the subfile access sequence in increasing order of these ranks is optimal in the sense of minimizing the worst case number of page accesses in processing $Q$. In our context, for $r_1$, $k_1$ and $k_2$ are given by $k_1 = \min\{n/B_1, c_1^k\}$ and $k_2 = \gamma_1$. Hence, $\rho(r_1)$ is obtained. In the similar manner, $\rho(r_2)$ can be obtained. □

In Lemma 3, the search cost is measured in terms of the worst case number of page accesses to sharply reduce the computation effort required for the cost evaluation. The algorithm for computing the expected number of page accesses for a random retrieval of tuples is recursive [137,133]. Therefore, the cost evaluation of each access sequence in terms of the expected number of page accesses is rather prohibitive to be embedded in a real-time query optimizer. Furthermore,
Table 2.4. Notations for the Sizes of the Relational Algebras

\[ \gamma^c = |\sigma_{\Lambda_{i \in SC}} F_i (r)| = \alpha, \text{ if } SC \neq \phi \]

\[ \gamma^k = |\sigma_{\Lambda_{i \in SK}} F_i (r)| = \alpha \]

\[ \gamma^{ck} = |\sigma_{\Lambda_{i \in SCUSK}} F_i (r)| = \beta(I) \]

\[ \gamma^i = |\sigma_{\Lambda_{i \in SCUSKUSN(I)}} F_i (r)| = |\text{view } i| = \beta(I) \]

\[ \gamma^1 = |\sigma_{\Lambda_{i \in SCUSKUSN(I)}} F_i (r)| = \beta(II), \text{ if } r(I) = r_1 \]

\[ \gamma^2 = |\sigma_{\Lambda_{i \in SCUSKUSN2}} F_i (r)| = \beta(II), \text{ if } r(I) = r_2 \]

\[ \gamma = |Q| = |\text{view } II| \]

Note: \( \gamma^{ck} \) becomes \( \gamma^c \) and \( \gamma^k \) if \( SK = \phi \) and \( SC = \phi \) respectively. Such equivalences given a particular query can occur among all the notations.

The worst case measure closely approximates the expected number of page accesses; especially when the blocking factor is greater than 20, which is most likely in segmented relational databases.\(^{(14)}\) Therefore, the chance of selecting a suboptimal access sequence due to the error in the worst case measure is believed to be negligible in query optimization.\(^{(15)}\)
II.6.2. **Processing subfiles**

After the sequence of subfile access is determined, the next step is to process $r(I)$ and $r(II)$. The database operations that are conditionally invoked in processing the respective subfiles are listed below (the first seven operations are also described in Chapter II.5):

**Operators:**

- **SCIND** Search the clustered index directory for the anchor points.
- **SIND** Search the non-clustered index directory.
- **INT** Intersect TID lists.
- **LINKP1** Given the TID's of the anchor points in $r_2$, compute the corresponding TID's in $r_1$.
- **LINKP2** Given the TID's of the anchor points in $r_1$, compute the corresponding TID's in $r_2$.
- **LINKF1** Given the TID's of all the tuples finally selected from $r_2$, compute the corresponding TID's in $r_1$.
- **LINKF2** Given the TID's of all the tuples finally selected from $r_1$, compute the corresponding TID's in $r_2$.
- **SEQP1** Sequentially retrieve tuples in a search range of $r_1$ defined by SCIND.
- **SEQP2** Sequentially retrieve tuples in a search range of $r_2$ defined by SCIND.
- **SCAN1** Sequentially scan the entire $r_1$.
- **SCAN2** Sequentially scan the entire $r_2$.
- **RAND1** Randomly retrieve tuples from $r_1$.
- **RAND2** Randomly retrieve tuples from $r_2$. 
As has been discussed in Chapter II.5, the index search operations SCIND, SIND and INT can be invoked only in processing r(I), while LINKP and LINKF are applicable only for processing r(II). The table access operation for each subfile can be one of the two alternatives -- random (RAND) or sequential -- and in the case of sequential, either SEQP or SCAN. Notice that SIND-INT-RAND, LINKF-RAND, and LINKP-SEQP are inseparable sequences of operations: the operations in the first sequence constitute the partial match retrieval in r(I); the latter two sequences realize the two different ways of direct linkage from r(I) to r(II).

Before we proceed, we first discuss Rules 9 and 10. These two rules are simple, but their results are indispensible in the rest of the production rules. Rule 9 states that if both SK and SN are empty sets, then random search can not be performed in either subfile (No RANDOM search). Thus, once Rule 9 is executed, the access choice in both subfiles becomes between SCAN and SEQP. Rule 10 says that the clustered index search SCIND is absolutely necessary if SC is not empty. If Rule 10 is executed, therefore, SEQP is enforced whenever the sequential access is chosen for a subfile.

Now we proceed to the production rules applying to r(I). Rules 11-16 are applicable when $I_1$ is true, i.e., $r(I)=r1$; Rules 23-28, when $I_2$ is true. Among these twelve rules, only one is executed for a particular query. If SK=∅, then there is no other choice but using the sequential search in processing r(I). Thus, SEQP or SCAN should be chosen according to the result of Rule 10. Hence, we have Rules 11, 12
and 23, 24. On the other hand, if \( SK^\phi \), then the random access is possible and Proposition 1 applies. If the sequential search is chosen by Proposition 1, then again, CIND in Rule 10 decides whether SEQP or SCAN should be used. These are reflected in Rules 13, 14 and 25, 26. The random access is chosen for searching \( r(I) \) only if \( SK^\phi \) and the selectivity falls below the trigger level. In this case, the partial match retrieval is enforced, with or without operation SCIND. These are represented as Rules 15, 16 and 27, 28.

The remaining rules, 17-22 and 29-34, are applied for determining an access path to \( r(II) \): the former rules are applicable when \( II_2 \) is true while the latter are when \( II_1 \) is true. Again, only one among these twelve rules is executed, when BOTH is true. Thus, the operators invoked in at most two rules, one chosen from Rules 11-16 and 23-28 and the other, if necessary, chosen from Rules 17-22 and 29-34, will constitute the access module for a query.

Access paths available for processing \( r(II) \) include SCAN, LINKP-SEQP and LINKF-RAND. The sequential search is enforced in processing \( r(II) \) if NRAND has been inferred. In this case, if CIND has not been created, SCAN is invoked. If CIND has been created, searching the entire \( r(II) \) can be avoided; instead, SEQP is enforced using the anchor points passed by LINKP. Rules 17, 18 and 29, 30 signal these situations.

The random search is applicable for searching \( r(II) \) if \( SK^\phi \) or \( SN(I)^\phi \). In this case, view \( i \) consists of tuples randomly placed on \( r(I) \). With the TID's of view \( i \) at hand, due to the direct linkage
between subfiles, the query optimizer can choose the random access method. If the selectivity is below the trigger level in \( r(II) \), all the TID's of view \( i \) are translated into TID's in \( r(II) \) first (LINKF) and then the random search is performed. Notice that if CIND has been created, the selectivity is \( \gamma^1/\gamma^c \) (or \( \gamma^2/\gamma^c \)); otherwise it is \( \gamma^1/n \) (or \( \gamma^2/n \)). These are rules 19, 20 and 31, 32. If, on the other hand, the selectivity is greater than the trigger level, then the sequential search is preferred to the random search. Again, CIND determines the way sequential search should be performed. If it is true, then LINKP-SEQP (Rules 21 and 33); otherwise, SCAN (Rules 22 and 34).
II.7 Examples

In this chapter, we illustrate query optimization by three examples. The problem solving mechanism used is the so called forward chaining: the antecedent of each rule is checked first and if it is true, then that rule is executed. All three examples are verified by the query optimizer program Appendix E.

Example 1:

Let \( R = \{1,2,3,4,5,6,7,8,9,10\} \), which is segmented into \( R_1 = \{1,3,5,7,9\} \) and \( R_2 = \{2,4,6,8,10\} \). Among the columns in \( R \), column 1 is the clustered index, i.e., \( C = \{1\} \), and columns 3, 4, 5 and 6 are non-clustered indexes, i.e., \( K = \{3,4,5,6\} \). Accordingly, \( N = \{2,7,8,9,10\} \). The cardinality of \( r \) is assumed to be \( n = 10,000 \). Assume the query has the projection attributes 1, 6 and 8, i.e., \( P = \{1,6,8\} \), and the selection attributes 1, 4, 7 and 10, i.e., \( S = \{1,4,7,10\} \).

The system first computes \( SC, SK, SN, SN_1, SN_2, P_1 \) and \( P_2 \). The system also estimates the sizes of relational algebras when they appear in the antecedent of a rule being checked. It also computes the rank of each subfile for Rules 7 and 8 when these rules are evaluated. The trigger level in each subfile, when necessary, is determined, using data in the system catalog, according to Proposition 2 in Chapter II.5.

102
For this example, the optimizer needs the following parameters: $\gamma^c$, $\gamma^c$, $\gamma^l$, $\gamma^l$, $r_1$, $r_2$, $\rho(r_1)$ and $\rho(r_2)$. Here we assume that $\gamma^c = 4,000$, $\gamma^c = 80$, $\gamma^l = 40$, $r_1 = 0.016$, $r_2 = 0.006$ and $\rho(r_1) > \rho(r_2)$. Then the system triggers Rules 3, 8, 10, 25 and 31, and generates a sequence of operators (SCIND SEQP2 LINKF1 RAND1). Thus, SCIND is applied to the clustered index directory first, followed by SEQP2 which sequentially searches a portion of $r_2$ and creates view i. The tuples contained in view i are mapped to $r_1$ by the direct subfile linkage operator LINKF1. Finally, the random search RAND1 is applied to $r_1$ to find the final response set. The reasons to choose this particular sequence can be found in the preconditions of Rules 3, 8, 10, 25 and 31. Notice that the selectivity is greater than the trigger level in $r_2$, but is smaller than the trigger level in $r_1$.

Example 2: If we change $r_1$ to 0.005 and $r_2$ to 0.023 in Example 1 and let all other parameters remain intact, then the query optimization plan becomes (SCIND SIND INT RAND2 LINKP1 SEQP1); the rules invoked are 3, 8, 10, 27 and 33.

Example 3: Assume that the relation, segmentation and indexes are the same as in the previous examples, but that a query has the projection attributes 4, 7 and 9, and the selection attributes 5 and 8. In this case, $\gamma^c = n$ and $\gamma^c = k$ because SC = $\phi$. Assume $\gamma^k = 200$, $\gamma^l = 200$, $\gamma^l = 20$, $r_1 = 0.024$, $r_2 = 0.011$. Then the rules invoked are 3, 6, 26 and 32 and the resultant plan is (SCAN2 LINKF1 RAND1).
II.8 Conclusions

In Part I of this dissertation, vertical segmentation has been shown to be an effective database structuring technique that can save database operating costs by 30-60%. The query processing strategies tailored to databases which incorporate vertical segmentation, however, have not been investigated in the literature yet. In Part II, we have developed a low-level optimizer for queries in conjunctive normal form which are processed on vertically segmented relations.

The optimizer takes into account the presence of both clustered and non-clustered indexes, and incorporates such factors as the optimal subfile access sequence and the optimal selection of an access method based on the size of a relational algebra applied. The search strategies underlying query optimization are complex and interrelated. These, however, are all combined into a rule-based system which provides an inference mechanism for generating the optimal choice of database operations to constitute the access module for each query. This rule-based query optimizer is simple to implement and we have provided a program in LISP in Appendix E.
PART III

OPTIMAL REORGANIZATION POLICIES
FOR STATIONARY AND EVOLUTIONARY DATABASES

105
III.1 Introduction

Modern databases frequently use file structures, such as indexed-sequential or hashed files, which employ overflow chaining for supporting random accesses. Such a file structure occasionally needs physical reorganizations in order to maintain the efficiency with which user requests on the database are processed (see [122] for a survey of database reorganization). The optimal reorganization policy minimizes the total operating cost including transaction and reorganization costs over a time horizon by specifying when to reorganize a file. This policy is determined based on the physical design of the file structure and the anticipated user requests on the file over a time horizon.

The system analyzed in this dynamic optimization problem comprises three basic components: the state dynamics of a file caused by update transactions such as insertions and deletions; cost structures of reorganization and different transactions requested on the file; and the decision structure. The state of a file is typically represented by the record population in the file. The operating cost depends on the file state at the time when the operation is performed. The reorganization decision is to be made at each transition of the file state balancing the trade-off between the extra cost of reorganization and its benefit in restoring the efficiency and hence reducing the cost of ensuing transactions.
III.1.1. The literature

Several authors looked into the optimal reorganization policy for file structures with overflow chaining, assuming that deleted records are not physically removed until the file is reorganized. Schneiderman [113], in Part I of his study, assumed that the search cost increases linearly in time over a finite horizon and derived algebraically the optimal fixed time interval between reorganizations. The reorganization cost was assumed to be fixed when the file size is stable, and to increase linearly in time when it grows. Tuel [127] considered variable length reorganization intervals for a growing file also under the assumption of linearly increasing search and reorganization costs. In Part II, Schneiderman [113] considered the case of random deterioration of the search cost and showed an algebraic computation of the optimal reorganization rule that is supposed to trigger reorganization when the search cost increases to a fixed level.

Yao, Das and Teory [138] incorporated file design issues into the study of reorganization policy by calculating the operating cost based on a detailed parameterization of the file structure, user transactions and storage device. Resultant search costs were nonlinear in time. They developed a heuristic criterion for dynamic reorganizations over an unknown time horizon assuming that frequencies of transactions per time unit are either fixed or a linear function of the file size. The heuristic criterion calls for reorganization as soon as the search cost savings due to reorganization become equal to the average reorganization cost per time unit.
Batory [10] also considered the integration of the reorganization policy and the physical file design problem. It was assumed that frequencies of transactions per time unit and hence the file growth rate are fixed over the planning period. He developed a method for predicting the evolution of a set of file parameter values based on the known growth rate of the file size. The values of such file descriptors were used to compute the expected operating cost at each discrete time point. A dynamic programming technique was employed to determine the optimal reorganization points and initial loading factors. A series of independent short-term optimizations were proposed using an updated projection of the frequencies in each consecutive planning period.

Mendelson and Yechiali [96] introduced a micro-level, i.e., record-by-record, development of the file state dynamics in the study of the reorganization policy. The operating cost was assumed to be a general increasing function of the number of records in the overflow area, ignoring complexities of cost structures. Thus, they restricted the system to include only the overflow area of a file. This led to an ergodicity of the state dynamics because the state, i.e., the number of records in the overflow area returns to the origin (zero) whenever a reorganization is enforced. Ergodicity of the system allowed them to apply the policy iteration method [72] to their solution technique. Under such conditions they proved that the optimal state-dependent policy is a control limit policy [45] that calls for reorganization as soon as the number of records in the system reaches a fixed value.
Larson [79], Heyman [67] and Cooper and Solomon [39] developed stochastic models of the file state dynamics and derived formulas for computing the expected passage time until the first overflow from a bucket, or the expected number of overflows per bucket in a given system time. Larson [79] and Cooper and Solomon [39] observed the file system at each change in the number of records in a file and modeled its dynamics as an instance of birth and death process, while Heyman [67] observed the system at each arrival of transactions and described the state dynamics as a discrete-time random walk model. Cooper and Solomon [39] provided a comparison of their work with the other two.

III.1.2. The approach

This work is the first study on the optimal reorganization policy that integrates the micro-level stochastic modeling of the file state dynamics with the detailed analysis of the internal structure of file design. Various restrictive assumptions employed in the past research are relaxed in this study. The policy space is not restricted by any prior definition of the decision structure. Therefore, the solution to this model will produce true optimality of the reorganization policy. The model observes the file system at the arrival of each transaction because an operating cost has to be captured for each transaction processed. The operating cost is formulated in terms of block accesses to secondary memory based on the physical file structure and the access path employed for implementing each operation.
We assume a large on-line database system where random retrievals and update transactions are quite frequent. Each type of transaction, such as random retrieval, sequential retrieval, insertion, deletion or modification, is assumed to have its own arrival rate with the interarrival times distributed according to an exponential distribution. Assuming that deleted records are not removed from the file until it is reorganized, the file state is depicted by two parameters: the number of non-deleted or active records (the logical file size) and the total number of records in the file (the physical occupancy of the file structure). The file state dynamics is derived from the assumed stochastic processes of transaction arrivals.

Our work is different from the past research, except [96], in that the file state dynamics is explicitly modeled as a stochastic process. In Batory's [10] deterministic model, for example, the fact that the arrival rate of transactions such as random retrieval may well be dependent on the file size at the time of each arrival was not considered. Our work is also different from [96] in that (i) cost structures of operations on the file are formulated at low-level details based on the physical file design, and (ii) the entire file, including both the prime and the overflow areas, is considered because costs of operations on the file are dependent on the structure of both areas.

When the entire file is considered, the stochastic process of the file state is non-ergodic if the logical file size shows an increasing trend over time. We call this case the evolutionary file system as contrasted with a stationary file system by which we signify the case
when the logical file size is stable as in a steady-state. A theoretical optimization model of reorganization policy is developed in this Part and efficient procedures for solving this model are presented for both cases.

The decision process in the evolutionary file system does not have the Markov property because a decision is dependent on its past history. The optimal control of such a non-Markovian process cannot be achieved with reasonable computational efforts if at each decision point we consider all possible values of state parameters associated with their respective probabilities. In order to make the optimization model computationally tractable, we use the expected values of state parameters. This enables us to formulate the model as a deterministic dynamic program which is solvable in polynomial time.

In the case of a stationary file system, due to the ergodicity of the state dynamics, the problem can be restricted to an analysis of a single reorganization cycle. This one-cycle optimization model is formulated as a stopping rule problem which is further reduced to an entrance-fee problem [18]. A linear programming formulation of the entrance-fee problem is suggested. It is proved that if the operating cost increases monotonically with the increase in file occupancy, the solution to this linear program becomes a control limit policy [45] similar to that derived in [96].

The rest of this Part is structured as follows. In Chapter III.2 the theoretical model and its solution procedures are developed for the
evolutionary and stationary file systems under generic assumptions on the physical file design and the stochastic process of transaction arrivals. In Chapter III.3 these theoretical results are validated by concrete examples of the stochastic process and the file design. The ISAM file, an IBM commercial product of order-preserving random access file, is used for the illustration. Chapter III.4 provides some concluding remarks.
III.2 A Theory of Optimal Reorganization Policies

III.2.1. Basic assumptions on the file structure and operations

In this section we describe a general structure of the file assumed in developing the theoretical model. The file is designed to support random accesses and comprises three conceptually separable areas: index area, prime data area and overflow data area. Indexed-sequential files and hashed files are typical examples of this structure. The prime area is composed of a number of equal-size partitions called buckets. A bucket contains an integral number of physical records or blocks, each block in turn containing an integral number (called the blocking factor) of logical records. A storage slot for a logical record is often called a subblock. A logical record is uniquely identified by the value of a certain field known as the (primary) key.

The index area or the directory has an address pointer to each bucket. The access method for the file has a function that relates a key value to a unique bucket. Therefore, given the user’s request of a certain record, the bucket containing that record can be accessed directly using the address indexed by the key value. A bucket is thus the maximum search space in a random access when there are no overflow
records. A fixed space for the overflow area is allocated at creation or reorganization time to accommodate new records arriving after their home bucket is filled up. The overflow area can be made up of several separate equal-size areas, e.g., a separate area for each bucket or for each cylinder, etc. In this chapter, however, a single independent overflow area is assumed without loss of generality. (The validity of this assumption will be discussed in Chapter III.3.)

Operations on the file are assumed to include random retrieval, sequential retrieval (in ascending order of key values), update and reorganization. The update transaction is further classified into insertion, deletion and modification. For convenience, batched updates are not considered (see [39] for a statistical treatment of batched updates). It is assumed that a deleted record is not physically removed from the file; instead, it is marked by placing a certain bit vector in the record. When a record is added to the file, if the home bucket is completely filled, one record (not necessarily the added one) is overflowed (see Section III.3.2 for further details). Overflow records originating from the same bucket are linked by address pointers to respective successors, and the first record in the overflow chain is linked to its home bucket. Since the overflow area is shared by many buckets and records are inserted on a first-come-first-served basis, the records overflowed from the same bucket are typically scattered around deteriorating the search time.

As records are added over time creating a long chain of overflows per bucket, the search or the serial processing time increases degrading
the file performance. A reorganization of the file structure can restore the performance by (i) removing deleted records reclaiming the space, and (ii) absorbing all overflow records into the restructured prime area. Of course, as soon as the overflow area is completely filled, reorganization is essential. Reorganization is performed by reading (unloading) the file sequentially and writing (reloading) all active records sequentially into the prime area of a new file structure, reclaiming the space used up by deleted records. During this process new indexes based on the changed distribution of key values are created by the reorganization program.

When a file is reorganized, buckets can be partially filled leaving spaces for new records. The portion of a bucket that is filled with data at reorganization time is called the initial loading factor. This factor affects the optimal reorganization policy; the lower it is, the longer an optimal reorganization interval would be, ceteris paribus. Other factors, such as the prime bucket size, blocking factors in the three areas and the size of overflow area, can also be determined when a file is reorganized. These factors, however, are assumed to be constant over the time horizon and determined exogenously to the model developed in this Part. Their effects on the optimal reorganization policy can thus be investigated through sensitivity analyses of our model.

III.2.2. Model formulation

In this section a generic optimization model for the reorganization policy is developed based on an assumed stochastic
process of file dynamics. This model can be used for the entire class of file structures defined in the previous section. The stochastic process of file dynamics developed below is quite general in that it can accommodate various hypotheses on the stochastic parameters.

In our model, a file is observed at each arrival of transactions. The time parameter of the model, \( t \in T = \{0, 1, 2, \ldots\} \) with 0 being the starting time of planning horizon, is integer-valued, increasing by one whenever a new transaction arrives. We use the term "system-time" to denote this time parameter in order to avoid confusion with real time \( t \) which is also set to be 0 at \( t=0 \). (Hereafter, when the term "time" is used alone, it will denote real time.) Let \( \lambda^k_r \) be the time-dependent arrival rate of transaction type \( k \) at time \( r \) where \( k \in K \) and \( K = \{s, r, a, d, m\} \) with the individual elements denoting sequential retrieval, random retrieval, addition, deletion and modification, respectively.

Additions and deletions result in changes in the size and the composition of record population of the file. The size of record population is measured by the number of filled subblocks in the file structure which can be dichotomized into active records and deleted records, or into records in the prime buckets and those overflowed. While the number of filled subblocks changes only with insertions, the number of active records changes with both insertions and deletions. These measures are major determinants of the operating cost. The efficiency of file operation also depends critically on the number of overflow records.
The state of a file at system-time \( t \) is summarized by a pair of parameters \((N_t, M_t)\) where \( N_t \) is the number of filled subblocks and \( M_t \) is the number of active records in the file. The process \((M_t)\) is purely stochastic, while \((N_t)\) is controllable by reorganization decisions. As a result of a reorganization at system-time \( t \), the number of overflow records becomes zero and \( N_t \) drops to \( M_t \) in the new file structure because \( N_t - M_t \) deleted records are physically removed during the reorganization. The expected number of overflow records at \( t \) can be expressed in terms of \( M_t \) [79,67,39], as will be shown in Chapter III.3.

Let \( p_t^k \) be the probability that a transaction arriving at \( t \) is of type \( k \). Then, \( p_t^{k} = \frac{\lambda}{\sum_{k \in K} \lambda} \), where \( r \) is the time when the \( t \)th transaction arrives, and \( \sum_{k \in K} p_r^k = 1 \). The process \((M_t)\) is a discrete-time Markov chain with a finite state space \( \Omega = \{0,1,\ldots\} \). The one-step transition probability at \( t \), \( p_{ij}^t(M) = \text{Pr}\{M_{t+1} = j | M_t = i\} \) is defined such that if \( i > 0 \), then \( p_{i+1,i}^t(M) = p_t^d \), \( p_{i-1,i}^t(M) = 1 - p_t^a \), \( p_{i,i}^t(M) = 1 - p_t^a - p_t^d \), and \( p_{0,1}^t(M) = 0 \) for \( j = 0, \) \( i - 1 \) or \( i + 1 \); if \( i = 0 \), then \( p_{0,1}^t(M) = 1 \), so that we have a reflecting barrier at \( M_t = 0 \). Consequently, \((M_t)\) is a random walk process (comparable to the process described by Eqs. (2.1) through (2.3) in [67]). Between reorganizations, the process \((N_t)\) is a monotonically increasing step function, i.e., \( p_{i,i+1}^t(N) = p_t^a \), \( p_{i+1,i}^t(N) = 1 - p_t^a \), and \( p_{i,j}^t(N) = 0 \) for \( j = i \) or \( i + 1 \).

Although a processing cost is incurred for every transaction, a reorganization will be considered only when the arriving transaction is an insertion. This is due to the fact that, in most file structures, between successive reorganizations the cost of processing a transaction as well as the cost of reorganization tend to increase with the increase
in \((N_t)\) (see cost formulas in Chapter III.3 for the case of an ISAM file), and \((N_t)\) increases only with an insertion. These costs would decrease or remain unchanged after any other type of transactions. Thus, the trade-off between the extra cost of reorganization and its benefit on the processing costs of ensuing transactions matters only at a point of insertion.

The system state is said to be in "stage" \(s\) if the \(s^{th}\) insertion has already occurred but the \((s+1)^{th}\) insertion has not occurred yet. The arrival rate \(\lambda_k^s\) for each transaction type \(k\) is assumed to be the same during a stage. Accordingly, the arrival rate is redefined as \(\lambda_s^k\), where \(k \in K\), for each stage \(s\). The duration of stage \(s\) is assumed to follow an exponential distribution with intensity \(\lambda_s^a\) and mean \(1/\lambda_s^a\). The probability of an arriving transaction being type \(k\) is also redefined to be \(p_k^s = \lambda_k^s / \sum_{k \in K} \lambda_k^s\), which is constant during a stage. During a stage \((M_t)\) may change while \((N_t)\) remains constant. \(M_s\) is, therefore, defined to be the number of active records at the time of an insertion, which is the first transaction that arrives during stage \(s\) according to the definition of stage.

Let \(C_s^k\) denote the cost of each transaction type \(k \in K\) in stage \(s\). A transaction cost in stage \(s\) is a function of two arguments \(N_s\) and \(M_s^o\), where \(s^o\) is the stage in which the latest reorganization was performed. The first argument \(N_s\) represents the physical occupancy of the file structure being used in stage \(s\) and the second argument \(M_s^o\) is a determinant of that file structure constructed at the latest.
reorganization in stage $s^o$. Chapter III.3 presents detailed formulations of the cost functions.

Let $f_s$ denote the expected cost of all transactions that arrive during stage $s$. It is calculated as the sum of the cost of an insertion and the expected cost of all transactions but an insertion that arrive during stage $s$. The number $X$ of all transactions arriving in stage $s$ is assumed to be distributed according to a geometric distribution, $\Pr(X=x)=p_s^a(1-p_s^a)^x$, $x=1,2,\ldots$, with mean $1/p_s^a$. Under this assumption, $f_s$ is given by

$$f_s(N_s,M_s^o)=C_s^a(N_s,M_s^o) + \frac{1}{p_s^a-1}\sum_{k,K,k=0} C^k_s(N_s,M_s^o) (p_s^k/(1-p_s^a))$$

where $C_0^a=0$ by the definition of stage.

The reorganization cost in stage $s$ is denoted by $F_s$ and is a function of $N_s$ and $M_s$. The expected opportunity cost of lost transactions during the reorganization should ideally be included in $F_s$. However, we assume that the reorganization is executed instantaneously in terms of the system-time so that this opportunity cost is negligible. This assumption is reasonable since in reality the file is usually reorganized during idle times of the database (e.g., at night). Given an optimal policy obtained under such an assumption, one could reorganize the file during the idle time which is closest to an optimal reorganization point.
Now we define the decision structure of the reorganization problem. The action space is defined to be a binary set $A = \{0, 1\}$ where 0 means to reorganize and 1 means not to reorganize. The decision structure is such that upon entering a stage, i.e., as soon as a new record arrives, at most two decision alternatives are available: If the capacity limit of the current file structure is exceeded by the insertion, action 0 has to be enforced; otherwise, either action 0 or 1 can be selected. The capacity limit of the current file structure is defined to be the space in terms of the number of subblocks allocated to the entire file - both prime and overflow areas - at the latest reorganization.

The effect of each action alternative is such that (i) if action 0 is taken in stage $s$, an extra cost $F_s(N_s, M_s)$ of reorganization is paid; the state instantaneously returns to $(M_s, M_s)$ and the expected cost $f_s(M_s, M_s)$ of transactions arriving in stage $s$ is paid observing transitions of the state from $(M_s, M_s)$ to $(M_s+1, M_s+1)$, and (ii) if action 1 is taken in stage $s$, $f_s(N_s, M_s)$ is paid and the state goes through transitions from $(N_s, M_s)$ to $(N_s+1, M_s+1)$.

Let $a_s \in A$ be the action chosen in stage $s$. The state-action sequence can be expressed as $\left( (N_0, M_0), a_0, (N_1, M_1), a_1, \ldots \right)$ and the reorganization policy $U$ can be defined such that $U = (u_0, u_1, \ldots)$ where $u_s$ is a function that relates the history $h_s$ of state-action sequence to the action being made in stage $s$, i.e., $a_s = u_s(h_s)$ and $h_s = \{(N_0, M_0), a_0, (N_1, M_1), a_1, \ldots a_{s-1}, (N_s, M_s)\}$. The decision process generally does not have the Markov property because a decision made in a stage depends on
the current file structure which was constructed sometime in the past.

The policy space $\Gamma_S$ for a finite time horizon with terminal stage $S$ is a Cartesian-product of $S$ copies of binary set $A$ and is a set of cardinality $2^S$.

Let $Z$ denote the total operating cost incurred over a finite time horizon. A reorganization policy $U^*$ is optimal if $E_U[Z] = \min E_U[Z]$. The expected total operating cost $E_U[Z]$ over an $S$-stage horizon if actions follow an optimal policy $U^*$ is given by

$$E_U[Z] = \sum_{s=0}^{S} \left\{ u_s^*(h_s) f_s(N_s, M_s) + (1-u_s^*(h_s)) (f_s(N_s, M_s) + f_s(M_s, M_s)) \right\}. \quad (3.2)$$

When the time horizon is infinite, i.e., $S \to \infty$, $E_U[Z]$ does not converge to a finite value for any $U$ and hence there does not exist a solution for $U^*$. In this case, instead of minimizing the total operating cost, the long-run expected operating cost per stage can be minimized, provided that this cost converges to a finite value in the limit.

Letting $W$ denote the operating cost per stage, $E_U[W]$ is expressed as

$$E_U[W] = \lim_{S \to \infty} \left\{ \frac{E_U[Z]}{S} \right\}. \quad (3.3)$$

In the next section we develop efficient procedures for solving $E_U[Z]$ and $E_U[W]$ for evolutionary and stationary file systems.
III.2.3. Solution procedures

III.2.3.1. Evolutionary file system

The evolutionary file system is a file system in which $M_s$ evolves stochastically with a positive growth rate, e.g., $\lambda^a_s > \lambda^d_s$ for all $s$. A finite time horizon is assumed in this case, for no file can grow forever because of the physical constraints of computer capacity. Also from a logical viewpoint, it is desirable for database control and efficiency to decompose a file into separate files based on the values of some field when the file size becomes excessively large. A time horizon can be determined in such a manner that the hypotheses about parameters of the stochastic process remain realistic over the horizon.

An exhaustive enumeration of the policy space $\Gamma_S$ to find an optimal policy $U^*$ has an exponential time complexity $O(2^S)$ where $S$ is the terminal stage, i.e., the total number of insertions arriving over the time horizon. A computationally tractable solution procedure is developed using the dynamic programming technique (see [14] for a survey of these techniques). According to the principle of optimality [13] Eq. (3.2) can be decomposed into a sequence of recursive functional equations $V_s$, often called Bellman equations, defined at every stage $s=0,1,\ldots,S$. $V_s$ is the optimal expected operating cost over the remaining period if the process started in stage $s$ and actions followed the optimal policy $U^*$. 
In Eq. (3.2), except for the initial condition $M_0 = N_0$, the state parameter $M_s$ is a stochastic variable and $N_s$ is dependent on the history of actions prior to stage $s$. Solving $V_s$ for all $s$ will become intractable even for a short-term horizon, if we formulate it in terms of the probability measures associated with all possible values of $M_s$. In order to make the solution procedure tractable, the expected value of $M_s$ is used for each stage $s$. $N_s$ can be expressed as $M_s + \eta$ where $\eta = s - s^*$, i.e., the number of records inserted since the latest reorganization until stage $s$. Using the expectation terms $\hat{N}_s = E[M_s|\lambda_s^a, \lambda_s^d]$ and $\tilde{N}_s = \hat{N}_s + \eta$, the equation for $V_s$ becomes

$$V_s(\hat{N}_s, \tilde{N}_s) = F_s(\hat{N}_s, \tilde{N}_s) + f_s(\hat{N}_s, \tilde{N}_s) + V_{s+1}(\hat{N}_s + 1, \tilde{N}_s), \text{ if } \tilde{N}_s + 1 \text{ exceeds the capacity limit in stage } s$$

$$= \min \left\{ f_s(\hat{N}_s, \tilde{N}_s) + V_{s+1}(\hat{N}_s + 1, \tilde{N}_s), \quad F_s(\hat{N}_s, \tilde{N}_s) + f_s(\hat{N}_s, \tilde{N}_s) + V_{s+1}(\hat{N}_s + 1, \tilde{N}_s) \right\}, \text{ otherwise.}$$

(3.4)

The boundary condition of $V_s$ at terminal stage $S$ is

$$V_S(\hat{N}_S, \tilde{N}_S) = F_S(\hat{N}_S, \tilde{N}_S) + f_S(\hat{N}_S, \tilde{N}_S), \text{ if } \tilde{N}_S + 1 \text{ exceeds the capacity limit}$$

$$= f_S(\hat{N}_S, \tilde{N}_S), \quad \text{otherwise.}$$

(3.5)

The file reorganization problem is very similar to the equipment replacement problem (see [38]). A reorganization is analogous to a replacement and the file structure constructed anew by a reorganization is analogous to a new equipment purchased to replace the old one. Using this analogy, it is appropriate to call $\eta$ the "age" of the current file structure. Given the relationship $\hat{N}_s = \tilde{N}_s - \eta$, Eqs. (3.4) and (3.5) can be
rewritten as a single-parameter deterministic model with \( \eta \) being used as the state parameter. Note that the expected value \( \hat{M}_s \) can be estimated as a constant given \( s \) and \( \eta \), once the stochastic process \( \{M_t\} \) is specified (see Section III.3.1). This single-parameter model is expressed in Eq. (3.6) with the boundary conditions specified in Eq. (3.7).

\[
V_s(\eta) = \min_{a_s(\eta) \in (0,1)} \left\{ a_s(\eta) f_s(\hat{M}_s, \eta, \hat{M}_s) + (1-a_s(\eta)) (f_s(\hat{M}_s, \eta, \hat{M}_s) + f_s(\hat{M}_s, \hat{M}_s)) + V_{s+1}(\psi(\eta)) \right\},
\]

where \( a_s(\eta) = 0 \) if \( \eta = \eta_s^\wedge \), for all \( s \) and \( \psi(\eta) = \eta - a_s(\eta) + 1 \), for all \( s \).

\[
V_s(\eta) = f_s(\hat{M}_s, \eta, \hat{M}_s) + f_s(\hat{M}_s, \hat{M}_s), \quad \text{if } \eta = \eta_s^\wedge
\]

\[
= f_s(\hat{M}_s, \eta, \hat{M}_s), \quad \text{otherwise} \quad (3.7)
\]

In the above equations, \( \psi(\eta) \) is the state transformation function and \( a_s(\eta) \) is the action taken in state \( \eta \) at stage \( s \). \( \eta_s^\wedge \) is defined to be the maximum possible age of the current file structure in stage \( s \). In other words, if the file structure being used in stage \( s \) had been constructed in stage \( s - \eta_s \), an insertion of a new record in stage \( s \) would cause the capacity limit of that file structure to be exceeded. Consequently, \( \eta_s^\wedge \) is a forced reorganization state in stage \( s \) and is related to the size of the space that is reserved for new records when the file structure is reorganized. This space comprises the overflow area and the empty room reserved in the prime area dictated by the initial loading factor. The computation of \( \eta_s^\wedge \) for the ISAM file will be shown as an example in Section III.3.2.
The minimum expected operating cost over the time horizon is given by \( V_0(0) \) which can be obtained by solving \( V_s(\eta) \) backwards starting from the terminal stage. The optimal policy is the sequence of actions that leads to this minimum cost. This optimal sequence of actions can be traced forward starting from stage 0 by applying the state transformation function \( \psi(\eta) \) in successive stages. The following theorem addresses time complexity of the dynamic program given in Eqs. (3.6) and (3.7).

**Theorem 3** The dynamic program given in Eqs. (3.6) and (3.7) for solving the optimal reorganization policy has a time complexity of \( O(S^2) \).

**Proof** The backward process of the dynamic programming algorithm for obtaining \( V_0(0) \) involves the computation of \( V_s(\eta) \) for \( \eta = 1, 2, \ldots, \eta_s \) in each stage \( s > s' \) such that in stage \( s' \) the file structure initially created at stage 0 is forced to be reorganized if there have been no voluntary reorganizations before stage \( s' \). In each stage \( s \leq s' \), \( \eta \) varies from 1 to \( s \). The computation of \( V_s \) for a given \( \eta \) requires evaluations of \( V_s(\eta) \) for the two cases when \( a_s(\eta) = 0 \) and \( a_s(\eta) = 1 \).

Measuring in terms of the number of cost evaluations (\( F \) and \( f \) in case \( a_s = 0 \), while only \( f \) in case \( a_s = 1 \)), the run time of the backward process is \( 2 \times \{ \sum_{s=1}^{s'} s + \sum_{s=s'+1}^{\eta_s} \} \). The forward process for tracing the optimal policy requires in each stage an evaluation of function \( \psi(\eta) \) and a comparison of two decision alternatives. Thus, its run time is far less than that of the backward process. Since \( \eta_s \) increases very slowly with \( s \) assuming a steady growth of the file size, the solution algorithm has a polynomial time-function in \( S \) which is dominated by \( 2 \times \sum_{s=1}^{S-1} s = S(S+1) \). □
The reader is referred to [38] for numerical examples of solution procedures of this kind in the context of replacement problems.

III.2.3.2. Stationary file system

The stationary file system signifies a file for which the stochastic process \( (M_t) \) is a martingale [74]. A typical case is when \( \lambda^a_s = \lambda^d_s \) and hence \( p^a_s = p^d_s \) for all \( s \), implying that \( (M_s) \) is a symmetric random walk. Since \( \bar{M}_s = N_0 \) for all \( s \) by the martingale property, it follows that \( \bar{N}_s = \bar{M}_s - \eta - N_0 + \eta \) and \( (N_s) \) is expected to return to the origin \( N_0 \) whenever a reorganization is performed. This implies that the file structure constructed at every reorganization is expected to be the same. \( (N_s) \) is thus a spatially homogeneous ergodic process with a reflecting barrier at \( \hat{N} \) which is the capacity limit of the file structure. \( \hat{N} \) is a forced reorganization state whose expected value is constant over time in this stationary system.

When we assume an infinite time horizon, the optimization model is given by Eq. (3.3). The expected costs \( f_s \) and \( F_s \) are expressed as univariate functions of a non-stochastic variable \( N_s \) because \( \bar{M}_s \) becomes a constant \( N_0 \) for all \( s \). The optimization model, therefore, becomes a single-state model with \( N_s \) being the state parameter. Reorganization policy \( U \) is defined to be a set of reorganization states including \( \hat{N} \). In this setting, Eq. (3.3) becomes
\begin{equation}
W_U = \lim_{s \to \infty} \frac{\sum_{s=0}^{S} \Psi(N_s)}{S}, \quad (3.8)
\end{equation}

where \( \Psi(N_s) = f_s(N_s) + f_s(N_0) \), if \( N_s \in U \)
\[
- f_s(N_s), \quad \text{if } N_s \notin U.
\]

The ergodic theorem [74] states that the long-run expected cost per stage (the right hand side of Eq. (3.8)) converges with probability 1 to \( W_U \) which is a constant determined by policy \( U \). Let us define a reorganization cycle to be the sequence of states that starts with \( N_0 \) and ends at the first visited state in set \( U \). Let \( \Phi_r \) and \( D_r \) be the cost and the duration of the \( r \)th cycle respectively, and let \( \Theta_r \) be the set of stages in that cycle. Then, \( \Phi_r = \sum_{s \in \Theta_r} \Psi(N_s) \) and \( D_r = |\Theta_r| \), the cardinality of set \( \Theta_r \). Since the cost \( \Phi_r \) and the duration \( D_r \) of different cycles are i.i.d. random variables with a finite mean, by the law of large numbers,
\[
\frac{\sum_{r=1}^{R} \Phi_r}{\sum_{r=1}^{R} D_r} \to \frac{E[U[\Phi_1]}}{E[U[D_1]]} \text{ with probability 1 as } n \to \infty.
\]
This leads to the equation \( W_U = \frac{E[U[\Phi_1]]}{E[U[D_1]]} \) where \( E[U[D_1]] = |\Theta_1| \). In this way the problem is reduced to one that involves only the first cycle.

Let \( \alpha^* = \max_{W_U = W^*} \Phi_r \) so that \( \max_{\alpha} \{ \alpha^*(\theta+1) - E[U[\Phi_1]] \} = 0 \), where \( \theta = |\Theta_1| - 1 \) is the last stage in the first cycle. Now let us denote the new objective function by \( Y \), i.e., \( Y = \alpha^*(\theta+1) - E[U[\Phi_1]] \). By definition,
\[
E[U[\Phi_1]] = \frac{\sum_{s=0}^{S=1} f_s(N_s) + f_\theta(N_\theta) + f_\theta(N_0)}{E[U[D_1]]} \text{ where } N_\theta \text{ is the first reorganization state that the system encounters. Substituting this expression for } E[U[\Phi_1]] \text{ in the equation for } Y, \text{ we obtain Eq. (3.9) as the objective function to be maximized.}
\]

\begin{equation}
\max_{\alpha^*} E[U[Y]] = \min_{\alpha^*} \left\{ \frac{\sum_{s=0}^{S=1} f_s(N_s) \cdot \alpha^*}{\sum_{s=0}^{S=1} f_s(N_s) \cdot \alpha^*} - \left\{ f_\theta(N_\theta) + f_\theta(N_0) - \alpha^* \right\} \right\}. \quad (3.9)
\end{equation}
Defining $G_s(N_s) = -F_s(N_s) - f_s(N_0) + \alpha^*$ and $g_s(N_s) = -f_s(N_s) - \alpha^*$, optimization model (3.9) can be interpreted as a single-parameter stopping rule problem defined as follows (see [26] for a general discussion on optimal stopping rules). Except at the forced stopping state $N$, we have two decision alternatives in state $N_s$: (i) Stop and receive the terminal payoff $G_s(N_s)$, or (ii) Pay an entrance-fee $g_s(N_s)$ and continue for at least one more stage. One-stage transition of the state is deterministic, i.e., $Pr\{N_{s+1} = i+1 | N_s = i\} = 1$ for $s = 0, 1, \ldots, \theta - 1$. The objective is to find $U^*_s$ such that $E_{U^*_s}[Y] = 0$.

Such a stopping rule problem can be further reduced to an equivalent entrance-fee problem [18] which by definition has $G_s(N_s) = 0$ for all $s$. Although one gets nothing for stopping in an entrance-fee problem, one still has an incentive to continue if $g_s$ is negative, viz., if one is "paid" to continue. The reduction is attained in our case by replacing $g_s$ by $g'_s$ expressed as

$$g'_s(N_s) = g_s(N_s) - \left\{ \sum_j Pr(N_{s+1} = j | N_s) G_{s+1}(j) - G(N_s) \right\}$$

$$= g_s(N_s) + G_s(N_s) - G_{s+1}(N_s)$$

$$= f_s(N_s) + (F_{s+1}(N_{s+1}) - F_s(N_s)) - \alpha^*. \quad (3.10)$$

The following theorem states the equivalence relation between the stopping rule problem and the entrance-fee problem.

**Theorem 4** The expected payoff $E_U[Y']$ in the entrance-fee problem using policy $U$ is given by $E_U[Y'] = \sum_{s=0}^{\theta} g'_s(N_s)$. If the policy $U$ is stable for both problems in the sense that $\lim_{s \to \infty} E_U[Y] = E_U[Y]$ and $\lim_{s \to \infty} E_U[Y']$
where \( U_s \) denotes a truncation of \( U \) at stage \( s \), then we have

\[
E_U[Y^*] = E_U[Y'] + G(N_0) = 0.
\]

Therefore, provided that both problems are stable, if \( U^* \) is a solution for one problem, it is also a solution for the other. (See [18] for the proof.)

Let us define \( V'(N_s) \) to be the optimal expected payoff obtained if the process started in state \( N_s \) and we followed the optimal policy \( U^* \) in the entrance-fee problem. Then we have

\[
V'(N_s) = \max \left\{ 0, -g'(N_s) + V'(N_{s+1}) \right\}, \text{ if } N_s < \hat{N} \\
= 0, \text{ if } N_s = \hat{N}.
\]

The optimal expected payoff of the entrance-fee problem is given by

\[
E_U[Y^*] = V'(N_0) = -G(N_0)
\]

and the optimal policy \( U^* \) is given by \( U^* = \{ \text{ all } N_s \text{ such that } V'(N_s) = 0 \} \).

By associating a variable \( x_s \) with each functional equation \( V'(N_s) \) for \( s = 0, 1, \ldots, \hat{N} - N_0 - 1 \), we can construct the following linear program which is equivalent to the entrance-fee problem. We call this linear program Problem (LP).

Minimize

\[
x_0
\]

subject to

\[
x_s - x_{s+1} \geq -g'(N_s), \quad s = 0, 1, \ldots, \hat{s} - 1
\]

\[
x_s \geq 0, \quad s = 0, 1, \ldots, \hat{s}
\]

where \( \hat{s} = \hat{N} - N_0 - 1 \). The equivalence relation between the entrance-fee problem and Problem (LP) is given in the following theorem.
Theorem 5 If \((x_0^*, x_1^*, \ldots, x_s^*)\) is the optimal solution to Problem (LP), then \(x_0^* - E_u^*[Y'] = -G(N_0) - F_0(N_0) + f_0(N_0) - \alpha^*\) and \(U^* = \{\text{all } N_s \text{ such that } x_s^* = 0 \text{ and } N\}\). (See [18] for the proof.)

The optimal reorganization rule is to reorganize the file structure when \(N_s\) first enters the optimal set \(U^*\). The optimal long-run expected cost per stage is given by \(\alpha^* = -x_0^* + F_0(N_0) + f_0(N_0)\).

In Problem (LP), the right-hand side of constraint (3.13), viz., \(-g_s'(N_s)\), includes \(\alpha^*\) which is unknown (see Eq. (3.10)). The value of \(\alpha^*\) can be found by solving Problem (LP)' which is obtained by replacing \(x_0^*\) by \(-\alpha^* + F_0(N_0) + f_0(N_0)\) in Problem (LP). Problem (LP) can be solved after Problem (LP)' is solved for \(\alpha^*\). Problem (LP)' is formulated as follows:

Maximize \(\alpha^*\) \hspace{1cm} (3.15)
subject to \(2\alpha^* + x_1 \leq 2f_0(N_0) + F_1(N_1)\) \hspace{1cm} (3.16)
\(\alpha^* + x_{s+1} - x_s \leq f_s(N_s) + F_{s+1}(N_{s+1}) - F_s(N_s), \ s=1,2,\ldots,\hat{s}-1\) \hspace{1cm} (3.17)
\(\alpha^* \geq 0 \text{ and } x_s \geq 0, \ s=1,2,\ldots,\hat{s}.\) \hspace{1cm} (3.18)

Both Problems (LP) and (LP)' have \(\hat{N}-N_0\) variables and \(\hat{N}-N_0-1\) constraints, where \(\hat{N}-N_0\) is the space in terms of the number of subblocks reserved for new records when the file is initially structured. Problems (LP) and (LP)' are in Class-P [76] and can be solved using any standard linear programming code.

Now we show that if transaction cost and reorganization cost are monotonically increasing functions of \(N_s\), the optimal reorganization
policy becomes a control limit policy [45] that calls for a re-
organization as soon as the system state \( N_s \) reaches a fixed integer \( N^* \).

**Theorem 6** If \( g' \) is a monotonically increasing function of \( N_s \), i.e., if
\[ g'(N_{s+1}) \geq g'(N_s) \]
for all \( s \), then there exists a fixed integer \( N^* \) such that the optimal policy is given by the set \( U^* = \{ \text{all } N_s \text{ such that } N_s \geq N^* \} \).
According to Eq. (3.10), if \( f \) and \( F \) increase monotonically with \( N_s \), so does \( g' \) and hence the optimal policy \( U^* \) applies.

**Proof** Associating \( y_0, y_1, \ldots, y_{s-1} \) with Constraint (3.13) for \( s=0, 1, \ldots, s-1 \) respectively, the dual of Problem (LP) is formulated as the following:

Maximize \[ \sum_{s=0}^{s-1} -g'(N_s) y_s \] (3.19)
subject to \[ y_0 \leq 1 \] (3.20)
\[ -y_s + y_{s+1} \leq 0, \quad s=0, 1, \ldots, s-2 \] (3.21)
\[ y_{s-1}^* \leq 0 \] (3.22)
\[ y_s \geq 0, \quad s=0, 1, \ldots, s-1. \] (3.23)

Constraints (20) through (23) can be rewritten as \( 0 \leq y_{s-1}^* \leq \ldots \leq y_1^* \leq y_0^* \leq 1 \).
Therefore, if \( g'_{s+1}(N_{s+1}) \geq g'(N_s) \) for all \( s \), then the optimal solution to the dual problem is \( y_s^* = 1 \) for all \( s \) such that \( g'(N_s) < 0 \) and \( y_s^* = 0 \) elsewhere. By the complementary slackness theorem [31], \( x_s^* = 0 \) whenever \( y_s^* y_{s-1}^* < 0 \) for \( s=0, 1, \ldots, s-2 \) and \( y_s^* = 1 \) for \( s=0 \). Let \( \bar{s} = \min( s \text{ such that } g'(N_s) > 0 ) \). Then \( x_s^* = 0 \) since \( y_s^* = 0 \) and \( y_{s-1}^* = 1 \). By the same theorem, \( x_s^* - x_{s+1}^* = -g'(N_s) \) whenever \( y_s^* > 0 \). Since \( y_s^* = 1 \) for all \( s < \bar{s} \), it follows that \( x_s^* - x_{s+1}^* = -g'(N_s) > 0 \) for all \( s < \bar{s} \) and hence \( x_s^* > 0 \) for all \( s \).
< \bar{s}. Thus \( x^*_s \) is the first \( x^*_s \) that has value zero and by Theorem 5, \( N^-_s \) is the first reorganization state that the system encounters. Therefore, the optimal policy is to reorganize as soon as \( N^s \) reaches \( N^-_s = N_0 + \bar{s} \), i.e., \( N^*_s = N_0 + \bar{s} \). □

Since \( x^*_0 = \frac{1}{s} \sum_{s=0}^{\bar{s}-1} [-g'(N_0+s)] \), using Theorem 5, \( \alpha^* \) is a solution to the equation \( F_0(N_0) + f_0(N_0) - \alpha^* = \sum_{s=0}^{\bar{s}-1} [-f_s(N_s) - F_{s+1}(N_{s+1}) + F_s(N_s) + \alpha^*] \). (Here \( [\cdot]^+ = \max(\cdot, 0) \).) This solution can also be obtained by solving Problem (LP)''. The optimal policy \( U^* \) in Theorem 6 can be translated to the rule saying that one should reorganize as soon as \( g'_s(N_s) \) turns positive from a negative value, i.e., as soon as the marginal operating cost \( f_s(N_s) + (F_{s+1}(N_{s+1}) - F_s(N_s)) \) per stage reaches or exceeds the optimal long-run expected cost \( \alpha^* \) per stage. This result is by and large the same as the optimal policy derived in [96], although our formal expression of the control limit policy differs from theirs because they modeled the system based solely on the overflow area, ignoring the effect of total record population on the operating cost and hence the optimal reorganization policy.
III.3 Applications of the Theory

III.3.1. Stochastic model of file state dynamics

In Section III.2.3 we used the expected value of the state parameter $M_s$ in order to make the model tractable. We estimate this expectation assuming typical instances of Markov process for transaction arrivals. In the case of an evolutionary file system, the state-dependent arrival rates for different types of transactions are specifically assumed such that $\lambda^a_i = \beta^a$ for all $i$, $\lambda^s_i = \beta^s$ for $i \geq 1$, and $\lambda^k_i = \beta^k$ for $i \geq 1$, for $k = d, r$ and $m$, where $\beta^k$ is a positive constant for all $k \in K$ and $i$ is the number of active records $M_r$ at the arrival time $r$ of a transaction. In the case of a stationary file system, $\lambda^k_r$ can be assumed to be constant over time for all $k \in K$ because $M_r$ is stochastically stationary.

The resultant process $(M_r)$ for the evolutionary file system is an instance of the birth and death process where birth and death rates are $\beta^a$ and $i \beta^d$ respectively, with $i = M_r$ at time $r$. This process $(M_r)$ implies that records are inserted to the file one-at-a-time according to a Poisson process with arrival rate $\beta^a$ and each of them stays active in the file for a residence time that is exponentially distributed with
intensity $\beta^d$. Thus, the evolutionary file system can be viewed as an
$M/M/\infty$ queueing system [62] where $M_r$ is analogous to the number of
customers in the system at $r$. It is possible to make inferences about
the insertion and deletion rates employed in such a birth and death
queueing model [136]. Therefore, the database administrator can decide
what model specification is appropriate for a given file system from
observing its behavior.

with the above assumptions the steady-state probability
distribution of $M_r$ is Poisson with parameter $\beta^a/\beta^d$. Thus the number of
active records in the file is $\beta^a/\beta^d$ in equilibrium. If the system starts
with $M_0 = \beta^a/\beta^d$, the expected value $\bar{M}_r$ is given by

$$\bar{M}_r = \frac{\beta^a}{\beta^d} \left( 1 - e^{-\beta^d r} \right) + M_0 e^{-\beta^d r}. \quad (3.24)$$

This equation implies an exponential approach to the equilibrium [74].
It is assumed that $M_0 < \beta^a/\beta^d$ in the case of an evolutionary file
system.

Since the expected passage time until the first arriving
transaction in stage $s$ is given by $s/\beta^a$, the quantity $\bar{M}_s$ defined in
Chapter III.2 can be approximately estimated by

$$\bar{M}_s = \frac{\beta^a}{\beta^d} \left( 1 - e^{-s\beta^d/\beta^a} \right) + M_0 e^{-s\beta^d/\beta^a}. \quad (3.25)$$
The probability $p^k_s$ of an arriving transaction being of type $k$ can be estimated; for instance, if $k=a$, then $p^a_s = \beta^a / (\bar{M}_s (\beta^d + \beta^r + \beta^m) + \beta^a + \beta^s)$. These approximate estimates of $\bar{M}_s$ and $p^k_s$ are to be used as arguments in Eqs. (3.1), (3.6), (3.7) and (3.10), and Problems (LP) and (LP)'. Note that in the case of a stationary file system, $\bar{M}_s = M_0 = N_0$ for all $s$ and $p^k_s$, $k \in K$, is constant over stages.

III.3.2. An analytic model of the file structure: the ISAM case

In what follows, the ISAM file, an order-preserving random access file marketed by IBM, is employed as an example to show how the terms defined in Section III.2.3 can be formulated based on a detailed description of the file structure. The terms to be derived include the operating costs $F_s$ and $G^k_s$, $k \in K$, and the factor $\eta_s$ imposed by the capacity limit of the file structure. For formulating the costs, it will be necessary to estimate the expected number of overflows at each stage.

III.3.2.1. Parameterization of the file structure

In this section the reader is assumed to be familiar with the ISAM file organization and its operations, or is referred to Claybrook [32] and its references. Here we only mention several assumptions that are employed to simplify cost function formulas without systematically biasing the results: Three index levels, i.e., master, cylinder and track indexes, are employed with the master index residing in main
Table 3.1. File Structure and Storage Device Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL</td>
<td>length in bytes of the primary key field</td>
</tr>
<tr>
<td>RL</td>
<td>length in bytes of a logical record</td>
</tr>
<tr>
<td>BFI</td>
<td>number of index entries in a block in the index area</td>
</tr>
<tr>
<td>BFP</td>
<td>blocking factor in the prime area</td>
</tr>
<tr>
<td>LF</td>
<td>initial loading factor</td>
</tr>
<tr>
<td>TV</td>
<td>number of tracks allocated for cylinder overflow</td>
</tr>
<tr>
<td>TY</td>
<td>number of tracks per cylinder</td>
</tr>
<tr>
<td>TL</td>
<td>length in bytes of a track</td>
</tr>
<tr>
<td>IBG</td>
<td>length in bytes of an inter-block gap</td>
</tr>
<tr>
<td>AT</td>
<td>length in bytes of the address of a track</td>
</tr>
<tr>
<td>AV</td>
<td>length in bytes of the link field in an overflow record</td>
</tr>
</tbody>
</table>
memory. The overflow area consists only of cylinder overflow areas, i.e., no independent overflow area is used. Deleted records are physically removed only during a reorganization.

The input parameters for the optimization models given in Eqs. (3.6) and (3.7) and Problems (LP) and (LP)' are listed in Table 3.1. The logical file parameters are assumed to be given and can typically be obtained from the database dictionary. The storage device parameters are also fixed for a specific hardware environment for the database. Here a movable head disk unit is assumed to be the storage device. The reorganization parameters can be determined by the database system administrator at reorganization times. However, for tractability of the optimization models, these parameters are also assumed to be constant over the entire time horizon. The effect of these parameters on the optimal policy can be examined through sensitivity analyses of our models.

Static descriptors of the internal file structure can be calculated as follows using the input parameters in Table 3.1. The values of these descriptors remain unchanged over the entire time horizon.

\[
\begin{align*}
PB &= TL/(IBG+KL+RL\times BFP) : \text{number of blocks in a prime bucket (track)} \\
PV &= TL/(IBG+KL+RL+AV) : \text{number of records (unblocked) in an overflow track} \\
PI &= TL/(IBG+(KL+AT)\times BFI) : \text{number of blocks in an index track} \\
TT &= (2\times(TY-TV)+1)/(PI\times BFI) : \text{portion of the first track of each cylinder that is allocated for track indexes}
\end{align*}
\]
From now on, we will deal only with the evolutionary file system.

The cost functions for a stationary file system can be obtained in a similar manner; all we need is to replace $\eta$, $\bar{M}_{s-\eta}$ and $\bar{M}_{s-\eta} + \eta$ by $s$, $M_0$ and $N_s (= N_0 + s)$ respectively in the expressions derived hereafter.

The following terms are dynamic descriptors of the internal file structure. They describe an instance of the file structure reorganized at some point in time (say, in stage $s-\eta$). Unlike the static descriptors mentioned above, the values of these terms remain unchanged only during a reorganization cycle (viz., in the cycle surrounding stage $s$).

\[ B_{s-\eta} = \bar{M}_{s-\eta} / (PB \times BFP \times LF) : \text{number of prime buckets in the file structure reorganized in stage } s-\eta \]

\[ Y_{s-\eta} = B_{s-\eta} / (TY - TV - TT) : \text{number of cylinders allocated for the file structure reorganized in stage } s-\eta \]

\[ I_{s-\eta} = Y_{s-\eta} / (PI \times BFI) : \text{number of tracks required for cylinder index storage in the file structure reorganized in stage } s-\eta \]

III.3.2.2. Overflow and capacity exhaustion problems

We assume that the probability of being hit by an inserted record is the same for each bucket, as was assumed in [79, 67, 39]. Keen and Lacy [75] showed that the probability that a key value randomly selected from the underlying distribution falls into a key value interval assigned to a bucket, is an i.i.d. random variable for all buckets. It follows that the expected probability of being hit by a new record is the same for each bucket, and that if the number of records loaded at
reorganization times is large enough, the actual value of that probability will be close to its expected value.

According to this assumption the arrival rate of records to each bucket is taken as an equal fraction \( \beta_s / B_{s-\eta} \), which is constant between reorganization times. Another consequence of this assumption is that the expected passage time until a forced reorganization will be the same regardless of whether or not the overflow area is split up into separate equal-size areas, e.g., the cylinder overflow areas in the ISAM file. The expected increase \( \mu_s \) of \( N_s \) in stage \( s \) is \( 1/B_{s-\eta} \) per bucket during the reorganization cycle surrounding stage \( s \). Let \( \omega_{s-\eta} \) denote the expected number of stages until the first overflow from a bucket after the latest reorganization in stage \( s-\eta \). Then \( \omega_{s-\eta} \) can be estimated by

\[
\omega_{s-\eta} = \frac{PB \times BFP \times (1-LF)}{\mu_{s-\eta}} = \frac{\tilde{M}_{s-\eta} (1-LF)}{LF}. \tag{3.26}
\]

This formula can be compared with Eq. 2.7 in [67] and Eq. 5 in [39].

Let \( \eta_s^* \) be such that if the file structure being used in stage \( s \) were older than \( \eta_s^* \), then there would be at least one overflow record in the file at stage \( s \); otherwise, no overflow record would exist. \( \eta_s^* \) is a solution to the following equation:

\[
\eta_s^* = \frac{\tilde{M}_{s-\eta_s^*} (1-LF)}{LF}. \tag{3.27}
\]
The expected number of overflow records per bucket when the file is \( \eta \) stages old at stage \( s \), denoted by \( \tilde{\sigma}_s(\eta) \), can be derived using \( \sigma_{s-\eta} \) as shown in Eq. (3.28). (These equations are comparable with Eq. 3.15 in [67]).

\[
\tilde{\sigma}_s(\eta) = \frac{(\eta - \omega_{s-\eta})/B_{s-\eta}}{P_{BFP} \times \left( LF \times (\tilde{M}_{s-\eta} + \eta)/\tilde{M}_{s-\eta} - 1 \right) }, \quad \text{if } \eta \geq \eta^* \\
= 0, \quad \text{if } \eta < \eta^*. \quad (3.28)
\]

Now we compute the oldest possible age \( \eta_s \) imposed by the capacity limit of the file structure being used in stage \( s \). The total number of subblocks available for new record insertions upon completion of a reorganization at stage \( s-\eta \) is \( \tilde{M}_{s-\eta} (1/LF - 1) + Y_{s-\eta} \times TV \times PV \). Since this number should equal to \( \eta_s \), the latter is a solution to the following equation:

\[
\eta_s = \tilde{M}_{s-\eta} \left\{ \frac{1}{LF} + TV \times PV / (P_{BFP} \times LF \times (TY - TV - TT)) \right\} - 1. \quad (3.29)
\]

III.3.2.3. Cost functions

Using the arguments of the previous sections, cost functions are formulated in terms of the number of block accesses to secondary memory required for processing the operations. The cost corresponding to the CPU time spent in processing the operations is ignored because this time is negligible compared to the I/O processing time. The file storage cost does not affect the optimal policy significantly and is ignored. The
cost functions can be formulated in terms of $s$ and $\eta$ because all their arguments become constants, given $s$ and $\eta$.

Since all records are logically sorted on the primary key value in an ISAM file, the cost $C_s^s(\eta)$ of sequentially retrieving all records in ascending order of key values includes reading the indexes and scanning every prime bucket followed, if necessary, by tracing its overflow chain. This cost is given by Eq. (3.30) assuming that an index block is kept in main memory until its entries are all utilized. (Note that overflow records are unblocked in the ISAM file.)

$$C_s^s(\eta) = (I_s - \eta + T T X Y_s - \eta) \times P I + (\hat{N}_s - \eta + \hat{O}_s(\eta) \times B_{s - \eta}) / B F P + \hat{O}_s(\eta) \times B_{s - \eta}.$$ (3.30)

The reorganization cost $F_s(\eta)$ consists of the cost of unloading the old file structure and of reloading the reorganized one. The unloading cost is the same as $C_s^s(\eta)$ and the reloading cost includes writing new indexes and prime buckets. Function $F$ is given by

$$F_s(\eta) = C_s^s(\eta) + (I_s^r + T T X Y_s) \times P I + B_s \times P B,$$ (3.31)

where $I_s$, $Y_s$ and $B_s$ describe the new file structure constructed at stage $s$. The cost $C_s^r(\eta)$ of randomly retrieving a record includes (i) one block access to search cylinder index, (ii) the average cost of scanning track index until the anchor point for the record's key value is found, and (iii) the average cost of scanning either the home bucket or its overflow chain until the record is found. Note that in the ISAM file if the record is in the overflow area, the access method directly accesses
the first record in the overflow chain without scanning the home bucket.

The cost is given by Eq. (3.32) assuming that one half of each search range is scanned on the average.

$$C^r_S(\eta) = 1 + 0.5 \left\{ \text{PIXTT} + \rho_s(\eta)(\bar{M}_s^- \eta + \eta)/B_s^- \eta - \bar{O}_s(\eta)) / \text{BFP} + (1 - \rho_s(\eta)) \bar{O}_s(\eta) \right\}, \quad (3.32)$$

where $\rho_s(\eta) = \text{PBxBFP}/(\text{PBxBFP} + \bar{O}_s(\eta))$ is the probability of a record being found in the prime bucket, i.e., the probability that the record's key value is less than or equal to the key value stored in the normal entry of the track index.

Since all records assigned to a bucket, whether in the prime area or in the overflow area, are to be logically sorted on key values in the ISAM file, if the added record must be inserted into the prime bucket, the records with higher key values in that bucket have to be rewritten being pushed to the right by one subblock. In this case if the home bucket has been filled completely prior to the insertion, the record with the highest key value in that bucket has to be moved to the overflow area. If the key value of the added record is larger than the highest key value in the completely filled home bucket, the record is placed directly in the overflow area, usually calling for an update of pointers in the overflow chain. The addition may also require updating the track index. The cost $C^a_S(\eta)$ of inserting a record is given by Eq. (3.33) which attempts to capture the average situation by assuming that the added record is placed in the middle of the home bucket or of the overflow chain.
The cost $C_s^d(\eta)$ of deleting a record and the cost $C_s^m(\eta)$ of modifying certain fields other than the key field of a record are computed as the random retrieval cost $C_s^r(\eta)$ plus one block access for rewriting the record. The cost of modifying the key field of a record is the same as the cost of deleting the record and then inserting a new record with the modified key value.

It is straightforward to compute optimal reorganization policies by applying the terms and cost functions derived in this section to the solution procedures developed in Section III.2.3.
III.4 Conclusions

A literature survey reveals that the solution procedures for optimal reorganization policies which can be readily applied in a stochastic environment to real-life databases with specific designs of component files are not available at present. In this Part we have presented a theoretical approach that integrates the micro-level stochastic model of file state dynamics and the cost structures of operations that are derived from a low-level analysis of the physical file design. We also developed polynomial-time procedures for solving such an integrated model for evolutionary as well as stationary databases. These procedures are general enough to be tailored to any specific file design within the broad class of file structures defined in Section III.2.1 and a specific stochastic model selected by the database administrator. The final algorithm for a given file will require a set of input parameters such as those given in Section III.3.2 that can be easily obtained at the system operator level. We believe that the solution based on an appropriate choice of stochastic parameters will give the database administrator a realistic guideline for reorganization decisions over a relatively long time horizon.

Our work can be easily modified to handle the case when deleted records are physically removed from the file. The file state dynamics can be then represented solely by \( M_L \) and the procedure for policy
optimization will accordingly become less complex. Incidentally, in that case, under our assumption of the equal arrival rate of insertions to each bucket, the stationary file system need not be given a serious consideration for reorganizations (see the steady-state analysis in [129]). Underlying mechanisms of the stochastic control revealed through our study have direct applications to similar issues such as database backups for recovery in case of failures [110,83], reorganizations of the differential file [119,2] and reorganizations of virtual storage files [94]. We hope that the work presented in this Part will encourage and facilitate the incorporation of effective algorithms for optimizing reorganization policies into database management systems.
PART IV

AN ADAPTIVE HEURISTIC FOR FILE MIGRATIONS

IN LOCAL AREA NETWORKS
IV.1 Introduction

With the advance of technology for small computers (workstations, minicomputer etc.) and data communication, distributed computing systems (DCSs) based on local area networks (LANs) became attractive alternatives to centralized computing systems based on mainframes. Universities, hospitals, and office buildings are examples suitable for LANs such as CSMA/CD and ring networks. Among many design and control issues in DCSs, the allocation and migration of files over a number of sites is considered to have great importance in improving the systems performance. In this Part we study the file migration policy applicable to LAN-based DCSs.

The file allocation problem (FAP) has been studied extensively [30,46,106]. The FAP distributes possibly replicated copies of files among different sites of a DCS to satisfy the system objective such as the minimal communication overhead and storage cost. The FAP can also take into account the presence of program files [97] and the system constraints such as response time, reliability, availability and storage capacity limit [85]. The FAP, however, assumes that users' file access frequencies over an entire planning horizon are known with certainty. The FMP is typically modeled as a linear or nonlinear mixed integer program. Such a static allocation could be justified in some DCSs based
on wide area networks (WANs). In these DCSs, rather strong locality of references exist; namely, a particular file is accessed more intensively from a few sites than others consistently throughout the planning horizon. The commercial bank where customers most often access files in their local branches is a typical example.

Dynamic allocation of files according to changes in users' file access rates, which is known as the file migration problem (FMP), has been studied by several authors. Levin and Morgan [81] assumed perfect information on access rates for successive discrete periods and presented a dynamic programming formulation of the FMP. Segall [114] and Segall and Sandell [115] modeled the file access pattern as a finite state Markov process and employed dynamic programming to obtain the optimal migration policy. Recently, adaptive heuristics have been developed for the FMP assuming only imperfect information on access rates. Levin [80] and Gavish and Sheng [58] proposed multi-period optimization of file allocations and estimated the expected access rates for the next period based on observations in the current period. They applied exponential smoothing procedures that would be sensitive to changes in trends.

The assumptions on users' file access pattern employed in all these studies, however, do not correspond to conditions in some real systems, especially in LAN-based DCSs [131]. The locality of references in LAN environment is usually much weaker than in WAN environment. Changes in the locality of references is rather fast so that extrapolation of a trend from the past access pattern cannot capture
transient variations. In such environments, significant remote file access overhead can be induced if file copies are not timely reallocated according to changes in access rates.

Empirical analyses of the file access pattern in local area environments have been performed by Revelle [107], Stritter [124], Smith [120] and Porcar [104]. The latter three examined the same file reference data which were collected over thirteen month period at the Stanford Linear Accelerator Center (SLAC). Smith [120] found that the distribution of interreference times of a file is more skewed than an exponential distribution and the expected time to next reference of a file is generally an increasing function of the time since the last reference. In a subsequent paper [121], Smith developed a long term policy for migrating files from disks to mass storage devices based on a data-derived model of file reference pattern.

Porcar [104] synthetically generated a LAN-based DCS by partitioning all users in the SLAC into a number of homogeneous groups, each representing a site in the synthetic DCS. He extended those policies developed for file migration in the memory hierarchy [43,105,44,121] to be applicable for file migration in LAN-based DCSs. Such policies were tested against the SLAC data using trace-driven simulation. These policies, however, were all demand policies that unconditionally transfer a file copy to a requesting site whenever it is remotely requested. Moreover, the storage limit at each site was not taken into consideration.
In this Part, we develop an adaptive file migration algorithm for LAN-based DCSs based on the observed behavior of users' file request pattern. A multi-access broadcast communication network is assumed and its properties are taken into account when the system cost is formulated. Distributed file migration decisions at individual sites are assumed allowing the site autonomy, and the primary copy method [128] is assumed for distributed file management and concurrency control. File migration decisions are considered to be made at fixed short time intervals and Bayesian approach is used for adaptive forecasting of file access rates over the next decision interval.

A file migration decision is to determine the set of remote files to be locally stored at a site for the next short time period considering the limited storage space at the site. We develop a dynamic optimization model for the file migration policy using Bellman approach. Based on this model, we derive a heuristic algorithm which can be used in multi-period independent migration decisions. It will be shown that this heuristic produces solutions almost identical to those of dynamic backward induction, if there is contention among remotely referenced files for the local storage space. We also investigate the effect of local storage expansion on the network traffic load.
IV.2 System Description

The DCS considered in this Part consists of a set of computing systems (sites) which are connected by a multi-access broadcast communication network. Examples of the latter are the CSMA/CD (e.g., Ethernet) and the token ring network. Each site has a processor and external storage devices. In such a LAN, effective distance of the network is not more than a few kilometers, the data rate is not higher than 10 Mbps and the number of sites that can be supported is at most 200. The logical properties of multi-access broadcast networks are as follows: (i) inter-site transmission time of a packet is site independent and hence the communication cost is proportional to the volume of data being transmitted; (ii) communication cost for updating a file is independent of the number of copies replicated in the network; and (iii) the message order is preserved in all sites so that synchronization and consistency control can be far more simplified than in point-to-point networks [131].

One can associate each permanent file in the DCS, either a user file or a system software, with a particular site where the file has been created or purchased. We call this site the primary site for that file. The file access pattern in LAN-based DCSs has been observed to have the following characteristics [104]: (i) a small portion of all
permanent files in a DCS are shared among more than one site, but these
shared files represent a large portion of the system's total I/O
activities; (ii) the primary site accounts for a large portion of all
accesses to the file and in most cases is the only one which updates the
file; and (iv) shared files are most often opened in read-only mode by
non-primary sites.

Based on these observations, we employ the primary copy method
[128] for distributed file management and concurrency control. For each
file in the DCS, its primary site keeps the file throughout its lifetime
and has all locking responsibility on that file. With this scheme, the
overhead of concurrency control such as locking message traffic is
minimal and hence is ignored. All sites other than the primary site will
be called the remote sites from the viewpoint of a file. On the other
hand, from the viewpoint of a particular site, a file is either called
the indigenous file if that site is the primary site of this file, or
called the remote file otherwise.

The organizational structure of most enterprises implementing LAN
for their distributed information systems supports decentralized
decision making among different sites. Furthermore, due to communication
overhead and transient nature of the file access pattern, it is
difficult to collect changes in access rates to a central site in real
time and make centralized global decisions for file migrations [131].
Therefore, file migration decisions are assumed to be made independently
at each site in distributed fashion allowing the site autonomy.
In such distributed decisions, it is difficult to allocate file update costs among the remote sites which maintain copies of the same file. Updating a packet of data in a file requires one broadcast on the network if the file is replicated at remote sites. To allocate the update propagation cost, a global information -- the degree of replication of a file in the network -- has to be maintained by the primary site and communicated to a remote site whenever the latter site makes a migration decision. To avoid such overhead, each site is assumed to pay the entire update communication cost regardless of the number of replicated copies in the network. In this case, the migration decision will be generally biased towards fewer replication of files in the network. But because the number of sites sharing a file is typically very small and remote file requests are mostly in read-only mode in real systems, such bias will not be significant.

Since we consider independent file migration decisions at individual sites, the DCS is described from a single site's viewpoint using the following parameters:

\( t \) : virtual time parameter that counts the total number of remote file references from the site, \( t=1,2,... \)

\( r_t \) : the arrival time of transaction \( t \)

\( \lambda \) : the intensity of the Poisson process governing the occurrence of all remote file requests at the site

\( \eta_t \) : the remote file referenced at virtual time \( t \)

\( r_t \) : the volume (in terms of the number of packets) of data retrieved from file \( \eta_t \)
n : index that counts the number of file migration decisions made over time, n=1,2,...

\( I_n \) : a set of remote files for the site during the \( n^{th} \) decision interval, i.e., between the \( n^{th} \) and \( (n+1)^{st} \) decisions

i : an individual file in set \( I_n \), i.e., \( i \in I_n \)

\( \ell_{i,n} \) : the size of file \( i \) in terms of the number of fixed size packets during the \( n^{th} \) decision interval

\( u_{i,n} \) : the volume (in terms of the number of packets) of data updated in file \( i \) during the \( n^{th} \) decision interval

Note that set \( I \) of remote files for a site can vary as files are created and scratched at their primary sites. Those files which are removed from disk to mass storage media due to infrequent usage at their primary sites are excluded from set \( I \). The following two variables represent the system state upon making the \( n^{th} \) migration decision:

\( s_n \) : the maximum disk space available for storing the copies of remote files; \( s_0 \), just prior to the first decision, is given.

\( F_n \) : a set of remote files whose copies are locally stored;

\( F_0 \), just prior to the first decision, is given.

Note that the constraint \( \sum_{i \in F_n} \ell_{i,n} \leq s_n \) must be satisfied.
IV.3 File Access Pattern and Decision Structure

File access pattern in local area environments [120,104] is summarized in the following: (i) the distribution of file usage frequency is very skewed, i.e., a few files are accessed a large number of times and most files are used very little; (ii) larger files are accessed more frequently, but the average fraction of a file read per open is smaller for larger files than smaller files; (iii) shared files are accessed more frequently and tend to be larger than non-shared files; (iv) if a file is in active use, it is likely to be referenced with high probability each day over a several day period; (vi) the interreference time of a file during the period of active use is very often less than an hour; (v) for a given short time period, a small number of files are repeatedly used and each of these files are accessed typically by a small number of users in a random order. Like such, there exists "time locality of references" for a file in local area environments. That is, a file is intensively used for a short period and then is seldom used for a while. We say a file is in "fashion" when it is in a period of active use. Fashions of a file can be considered to arrive according to a Poisson process, but the FMP model developed in this Part does not require a specific stochastic process for fashion arrivals.
The observed pattern implies the following for the FMP: (i) file migration decisions should be made with short time intervals in order to reflect rapid changes in fashions -- intuitively, if a file becomes in fashion due to frequent requests from a certain remote site, it should be cost effective to transfer a copy to that site as early in the current fashion as possible; (ii) each migration decision should depend on recent history of file references for predicting the probability of each file being referenced in the next short time period. The size of the recent history, called the window size, should also be short enough not to filter out a fashion of a file.

Therefore, file migration decisions are considered to be made at fixed short time intervals of length $\delta$ (in clock time) based on the observed references during the last period of the same length, i.e., the window. The decision interval or the window size $\delta$ should be much shorter than both the average duration of a fashion and the average interarrival time of fashions. Practically, $\delta$ can be four working hours for instance. The overall level of system activity generally shows a very strong seasonality with the hour of a day and the day of a week. It is thus desirable to postpone a next migration decision during non-working hours until the system resumes normal activity. In order to deseasonalize the time of a day and the day of a week effect, one can map file references, if any, during non-working hours onto the first working hour that follows. By doing this, the calendar time can be mapped into a series of consecutive decision intervals.
Let $p_{i,t}$ denote the probability of $\eta_t$ being file $i$. The prior distribution of $p_{i,t}$ is assumed to be a beta distribution with parameters $\alpha_i$ and $\beta_i$ regardless of $t$. For most files those parameters will both take values in open interval $(0,1)$ with the beta distribution being bowl-shaped on $[0,1]$ domain. This means $p_{i,t}$ is most likely to be either close zero when file $i$ is out of fashion, or significantly larger than zero when it is in fashion.

Define $\xi_{i,t}$ such that $\xi_{i,t}=1$ if $\eta_t=i$; $\xi_{i,t}=0$ otherwise. Then $(\xi_{i,t}, t=1,2,\ldots)$ is a random sample from a Bernoulli distribution with the parameter being distributed according to the beta distribution with parameters $\alpha_i$ and $\beta_i$. Therefore, the posterior distribution of $p_{i,t}$ when $\sum_{j=t-m}^{t-1} \xi_{i,j} = k$, i.e., when $k$ references of file $i$ have been observed from a reference string of size $m$, is also a beta distribution with parameters $\alpha+k$ and $\beta+m-k$. Here, the beta distribution is a conjugate prior distribution [41] for the sample $(\xi_{i,t}, t=1,2,\ldots)$.

Define $\Delta_n$ to be a file reference string observed during the window preceding the $n$th decision interval, i.e., $\Delta_n = \{ \eta_t : \tau_t \in [(n-1)\delta, n\delta) \}$. Let $k$ be the number of file $i$ occurrences in the file reference string $\Delta_n$ with length $m$. The posterior probability of $p_{i,t}$ during the $n$th decision interval, denoted by $\hat{p}_{i,n}$, is assumed to be constant for all $\eta_t \in \Delta_{n+1}$ and is given by

$$\hat{p}_{i,n} = \Pr(\eta_t=i, \eta_t \in \Delta_{n+1})$$

$$= \frac{(a_i+k)/(a_i+\beta_i+m)}{(k/m) k/(a_i+\beta_i+m) + (a_i/(\alpha_i+\beta_i)) (\alpha_i+\beta_i)/(\alpha_i+\beta_i+m)}.$$  (4.1)
It is notable that \( \hat{p}_{i,n} \) is a weighted average of the maximum likelihood estimate \( k/m \) of \( p_{i,t} \) and the mean \( \alpha_i/(\alpha_i + \beta_i) \) of the prior distribution of \( p_{i,t} \). Moreover, the respective weights are \( m/(\alpha_i + \beta_i + m) \) and \( (\alpha_i + \beta_i)/(\alpha_i + \beta_i + m) \). For most files with \( 0 < \alpha_i, \beta_i < 1 \), very small weight is given to prior belief on the distribution of \( p_{i,t} \). This property allows the adaptive policy based on the posterior distribution to fully reflect the information obtained from recent observations.

At each discrete time with fixed interval \( \delta \), a file migration decision \( d_n \) is to determine the state variables \( (F_n, s_n) \) based on information collected during the window. A finite horizon file migration policy \( \Phi \) is defined such that \( \Phi = (\phi_1, \ldots, \phi_N) \), where \( \phi_n, n=1,2,\ldots,N \), is a function that relates the file access history during the window and the previous state \( (F_{n-1}, s_{n-1}) \) to the decision \( d_n \).

At the \( n \)th decision, the set \( F_{n-1} \) is reconfigured into the set \( F_n \) as a result of transferring any number of new remote file copies from respective primary sites and deleting any number of existing copies of remote files at the local site. Some remote file copies in the set \( F_{n-1} \) may, of course, remain in the set \( F_n \). Such file migration decision is, however, subject to the storage capacity constraint imposed by \( s_n \). The state variable \( s_n \) may change without any control action, because it is the residual space besides the space allocated for all indigenous files and the latter space is variable over time. A control action on \( s_n \), if taken, is to expand the total storage capacity at the site by acquiring additional disk spindles. Since disk storage units are sold only in
several discrete sizes, $s_n$ will increase dramatically by a large amount upon such an action.

In summary, a decision $d_n$ is to determine (i) the set of remote files $F_n$ whose copies are to be kept locally during the $n$th decision interval and (ii) whether and how much to expand the storage capacity by installing new disk spindles. In this Part, the former decisions are optimized with the storage capacity known at each decision point. The latter decisions are examined in turn and we present the necessary condition for a storage expansion and its effect on the system cost.
The system cost is measured in terms of the number of data packets that have to be broadcast on the communication medium. The cost of enforcing a file migration decision includes (i) the cost of transferring copies of remote files in the set $F_n - F_{n-1}$, and (ii) the cost of deleting copies of remote files in the set $F_{n-1} - F_n$. Since the cost of deleting a file copy is negligible, the expected cost of enforcing the $n$th decision is given by

$$MC_n = \sum_{i \in (F_n - F_{n-1})} \bar{I}_{i,n}$$

(4.2)

Note that a bar over an argument stands for the expected value conditional on observations during the window throughout this Part.

During the $n$th decision interval, any remote file that is not locally available would require remote file open and access if it is referenced from the local site. Hence the remote processing cost $RC_n$ during the $n$th decision interval is

$$RC_n = \sum_{t: \eta_t \in (I_n - F_n), \eta_t \in \Delta_{n+1}} \bar{r}_t$$

(4.3)
On the other hand, storing a remote file copy at the local site entails the cost of maintaining the copy, i.e., the storage cost and the update propagation cost. The former, however, is accounted for by the storage capacity constraint and only the latter is considered to contribute to the system cost that has to be minimized. In such a model as is developed in this Part, the secondary storage still bears opportunity cost when there is contention among remotely referenced files for the available storage space. Consequently, the cost $UC_n$ of maintaining set $F_n$ of remote files during the $n$th decision interval is

$$UC_n = \sum_{i \in F_n} \hat{u}_{i,n}$$  \hspace{1cm} (4.4)

The total system cost $TC_n$ is the total traffic load required for processing all file access requests from the site during the $n$th decision interval and is the sum of $MC_n$, $RC_n$ and $UC_n$.

If we denote the conditional expectation under a given policy $\Phi$ by $E_\Phi$, our objective is to obtain a policy $\Phi^*=(\phi^*,\ldots,\phi^*)$ that leads to

$$V_0 = E_{\Phi^*}(\sum_{n=1}^N T\bar{C}) = \min_{\Phi} E_{\Phi}(\sum_{n=1}^N T\bar{C}), \text{ s.t. } \sum_{i \in F_n} \bar{u}_{i,n} \leq s_n, V_n.$$  \hspace{1cm} (4.5)

By the principle of optimality, $V_0$ and $\Phi^*$ can be obtained by solving the following family of conditional Bellman equations recursively backwards from the $N$th decision:

$$V_n(F_{n-1}) = E_{\Phi^*}(\sum_{j=n}^N E T\bar{C}_j), \text{ s.t. } \sum_{i \in F_j} \bar{u}_{i,n} \leq s_j, \text{ for } j=n,\ldots,N$$
\[ \min_{\phi_n} \left( \tilde{\mathcal{C}}_n + \mathcal{V}_{n+1}(F_n) \right), \text{ s.t. } \sum_{i \in F_n} \beta_{i,n} \leq s_n \]  \quad (4.6)

The boundary condition for these recursive equations is given by

\[ \mathcal{V}_N(F_{N-1}) = \tilde{\mathcal{C}}_N, \text{ where } \phi_N \text{ requires } F_N = F_{N-1}. \]  \quad (4.7)
IV.5 Adaptive File Migration Policy

IV.5.1. Adaptive policy under no storage capacity constraint

Dynamic optimization model given in Eqs. (4.6) and (4.7) in the previous chapter are not solvable, because users' behavior of accessing a particular file in local area environments cannot be modeled as a stochastic process whose statistical properties are known to us. In order to derive an adaptive file migration heuristic that still meets the objective defined in Eq. (4.5), we first consider the case when there is no limit on the storage space available for remote file copies. Without storage constraint, file migration decisions can be made independently for individual files. The decision becomes one that determines for each remote file whether or not to store its copy locally during a decision interval. If we define a binary decision variable $x_{i,n}$ for each file $i$ such that $x_{i,n} = 1$ if $i \in F_n$ and 0 otherwise, decision $\phi_n$ becomes a set of mutually independent decisions $x_{i,n}$ for all $i \in I_n$. The contribution of file $i$ to the total system cost $T_{C_n}$ is independent of all other remote files and is summarized in Table 4.1.
Table 4.1. The Contribution of File $i$ to the System Cost $T_C_n$

<table>
<thead>
<tr>
<th></th>
<th>$U_C_n$</th>
<th>$M_C_n$</th>
<th>$R_C_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $i \in F_{n-1}$, $i \in F_n$</td>
<td>$\hat{u}_{i,n}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>if $i \in F_{n-1}$, $i \notin F_n$</td>
<td>0</td>
<td>0</td>
<td>$\sum_{\eta_t=i, \eta_t \in A_{n+1}} \hat{r}_t$</td>
</tr>
<tr>
<td>if $i \notin F_{n-1}$, $i \in F_n$</td>
<td>$\hat{u}_{i,n}$</td>
<td>$\hat{\lambda}_{i,n}$</td>
<td>0</td>
</tr>
<tr>
<td>if $i \notin F_{n-1}$, $i \notin F_n$</td>
<td>0</td>
<td>0</td>
<td>$\sum_{\eta_t=i, \eta_t \in A_{n+1}} \hat{r}_t$</td>
</tr>
</tbody>
</table>

In the table, the cost of remotely processing file $i$ during the $n$th decision interval, i.e., $\sum_{\eta_t=i, \eta_t \in A_{n+1}} \hat{r}_t$ can be replaced by $\lambda \delta \hat{p}_{i,n}$ $\hat{r}_{i,n}$. Here $\hat{r}_{i,n}$ is the expected amount of data traffic per remote access to file $i$ during the $n$th decision interval and can be estimated to be the corresponding average observed during the window. In the same manner, $\hat{u}_{i,n}$ is estimated as the total volume of data actually updated in file $i$ during the window.

Without storage constraint, therefore, Eqs. (4.5), (4.6) and (4.7) can be rewritten as the following Eqs. (4.8), (4.9) and (4.10) respectively:

$$V_0 = \sum_{n=1}^{N} \sum_{i \in F_n} \min_{x_{i,n} \in (0, 1)} \{ (1-x_{i,n}) (1-x_{i,n} - 1)x_{i,n} \hat{\lambda}_{i,n} + (1-x_{i,n}) \lambda \delta \hat{p}_{i,n} \hat{r}_{i,n} + x_{i,n} \hat{u}_{i,n} \}$$  \hspace{1cm} (4.8)
In the above equations, $V^i_n$ is a recursive function defined for an individual file $i$ and is the minimum cost of processing file $i$ requests over the planning horizon starting from the $n$th decision. In Eq. (4.10), $\phi_N$ requires $x_{i,N} = x_{i,N-1}$ for all $i \in I_N$.

The above formulation implies that the optimal adaptive policy would be to enforce the following IF-THEN rule for each file:

1. **given $x_{i,n-1} = 0$,** if $\lambda \hat{p}_i \hat{r}_i i_{i,n-1} - \hat{u}_i i_{i,n} \geq V^i_{n+1}(1) - V^i_{n+1}(0)$, then $x^*_i i_{i,n} = 1$; otherwise, $x^*_i i_{i,n} = 0$; and

2. **given $x_{i,n-1} = 1$,** if $\lambda \hat{p}_i \hat{r}_i i_{i,n-1} - \hat{u}_i i_{i,n} \geq V^i_{n+1}(1) - V^i_{n+1}(0)$, then $x^*_i i_{i,n} = 1$; otherwise, $x^*_i i_{i,n} = 0$.

In the IF parts of the rule, the left hand side of $\geq$ can be estimated based on observations during the window. The right hand side, i.e., $V^i_{n+1}(1) - V^i_{n+1}(0)$, on the other hand, is unknown at the decision time. It will, however, be either 0, if $x_{i,n+1} = 0$, or $-\hat{r}_i i_{i,n+1}$, if $x_{i,n+1} = 1$. 

\[
\sum_{i \in I_n} V^i_i(x_{i,n-1}) = \sum_{i \in I_n} \min_{x_i \in \{0, 1\}} \left\{ (1-x_i i_{i,n-1}) x_{i,n} + (1-x_i i_{i,n}) \lambda \hat{p}_i \hat{r}_i i_{i,n} + x_i i_{i,n} \hat{u}_i i_{i,n} + V^i_{n+1}(x_{i,n}) \right\} \\
\sum_{i \in I_N} v^i_i(x_{i,N-1}, V_{i \in I_N}) = \sum_{i \in I_N} \left\{ (1-x_i i_{i,N-1}) \lambda \hat{p}_i i_{i,N} \hat{r}_i i_{i,N} + x_i i_{i,N-1} \hat{u}_i i_{i,N-1} \right\}
\]
In order to replace the uncertain variable $V_{n+1}^i(l) - V_{n+1}^i(0)$ by a constant, we define $\theta$ to be the probability of $x_{i,n}$ being 1. Of course $\theta$ is not measurable, but we assign a constant value to $\theta$ for the time being. Given some chosen value of $\theta$, the proxy measure of $V_{n+1}^i(l) - V_{n+1}^i(0)$ would be $-\theta \lambda_{i,n+1}$. For example, if $\theta$ is set to 1, the policy becomes the demand migration policy which transfers a copy of a remote file if the file is referenced at least once during the window. In the next section, it will be shown that the value of $\theta$ has negligible effect on the optimal migration policy when there is contention for the local storage space among remotely referenced files.

IV.5.2. Adaptive policy under storage capacity constraint

With a fixed limit on the storage space for remote file copies, one has to optimally select the remote files to be stored locally. Let us define $c_{i,n}$ to be the savings in the system cost by having a copy of file $i$ at the site during the $n$th decision interval. Estimating $\lambda_{i,n+1}$ to be the same as $\lambda_{i,n}$, we have

$$c_{i,n} = \lambda \delta p_{i,n} \bar{r}_{i,n} - (1-\theta) \lambda_{i,n} \bar{u}_{i,n}, \text{ if } i \notin F_{n-1} \quad (4.11)$$

$$c_{i,n} = \lambda \delta p_{i,n} \bar{r}_{i,n} + \theta \lambda_{i,n} \bar{u}_{i,n}, \text{ if } i \in F_{n-1} \quad (4.12)$$

In the sequel, we drop the subscript $n$ because the file migration rule is the same at all decision points. Define a set $I^+$ of remote files such that $I^+ = \{i: c_i > 0\}$. If $\Sigma_{i \in I^+} \lambda_{i} \leq s$, then the optimal file assignment $F$
during a decision interval is to have a copy of every file in set $I^+$. The problem of selecting remote files to be included in set $F$ when $\sum_{i \in I^+} \hat{\lambda}_i > s$ is formulated as the following 0-1 knapsack problem, called Problem (P):

$$\max \sum_{i \in I^+} c_i x_i \quad \text{s.t.} \quad \sum_{i \in I^+} \hat{\lambda}_i x_i \leq s, \quad x_i \in \{0, 1\} \forall i \in I^+. \quad (P)$$

Problem (P) is NP-complete but can be solved for its optimality in pseudo-polynomial time [55]. A state-of-the-art algorithm for solving (P) for the optimal solution can be found in Nauss [98] and Balas and Zemel [9]. Here, we employ a simple heuristic, called Algorithm (H), that selects remote files solely based on the efficiency ratio $b_i = c_i / \hat{\lambda}_i$.

Algorithm (H)

1. Order remote files $i \in I^+$ by decreasing $b_i$ and index these files so that $b_1 \geq b_2 \geq \ldots \geq b_{|I^+|}$.
2. Let $k$ denote the file index such that $k = \sup\{m: \sum_{i=1}^{m} \hat{\lambda}_i \leq s\}$.
   Let $X^*$ be a solution such that $x_i^* = 1$ for $i = 1, 2, \ldots, k$ and $x_i^* = 0$ elsewhere. If $\sum_{i=1}^{k} \hat{\lambda}_i = s$, then $X^*$ is optimal and goto 5.
   Otherwise, set $a = s - \sum_{i=1}^{k} \hat{\lambda}_i$ and $j = k+1$.
4. If $\hat{\lambda}_j \leq a$, set $x_j^* = 1$. Set $a = a - \hat{\lambda}_j$. If $a > 0$, goto 3.
5. Terminate
Algorithm (H) puts extra remote files in the slack \( \alpha \) in decreasing order of \( b_i \) until no more files fit. Since the storage capacity \( s \) or the RHS of the knapsack constraint is usually much greater than individual file sizes \( \lambda_i \) or weights in the knapsack constraint, the heuristic value obtained by Algorithm (H) would be very close to the optimal value of Problem (P). In Problem (P), however, we still have the proxy parameter \( \theta \) whose value it is difficult to predict.

If we let \( c'_i = c_i - \theta \lambda_i \), the objective function of (P) can be rewritten as \( \sum_{i \in I^+} (c'_i + \theta \lambda_i) x_i \) and \( b_i \) becomes \( c'_i / \lambda_i + \theta \). Applying Algorithm (H), the heuristic solution of (P) would be the same as that of the following problem, called problem (P'), provided \( \sum_{i \in I'} \lambda_i > s \).

Here \( I' \) denote a set of remote files such that \( I' = \{ i : c'_i > 0 \} \).

\[
\text{Max} \sum_{i \in I'} c'_i x_i \quad \text{s.t.} \quad \sum_{i \in I'} \lambda_i x_i \leq s, \quad x_i \in \{0,1\} \quad \forall i \in I'.
\]

(P')

Note that Problem (P') no longer includes parameter \( \theta \). Thus, the adaptive file migration policy can be implemented without need for guessing \( \theta \) if the condition \( \sum_{i \in I'} \lambda_i > s \) is met.

Finally, we analyze the impact of the local storage capacity on the system cost. The following theorem gives the range for the elasticity of the heuristic value of (P) to the storage space \( s \).

**Theorem 7** Let \( F \) and \( V(s, I^+) \) denote the heuristic selection of remote files from set \( I^+ \) given the storage limit \( s \) and the corresponding system
cost during the decision interval, respectively. Also let \( \bar{F} \) be the size of the largest file in set \( I^+ \). Then, provided \( \Sigma_{i \in I^+} \bar{F}_i > s + \Delta s \), we have
\[
\frac{V(\Delta s, I^+ - F)}{V(s, I^+)} \leq \frac{[V(s+\Delta s, I^+)-V(s, I^+)]}{V(s, I^+)} \leq \frac{(\Delta s + \bar{F})}{(s - \bar{F})}. \tag{4.13}
\]

**Proof**

(1) First, the proof of \( \frac{V(\Delta s, I^+ - F)}{V(s, I^+)} \leq \frac{[V(s+\Delta s)-V(s)]}{V(s)} \) is straightforward, since \( V(s+\Delta s, I^+) \geq V(s, I^+)+V(\Delta s, I^+ - F) \). □

(2) We now prove \( \frac{[V(s+\Delta s)-V(s)]}{V(s)} \leq \frac{(\Delta s + \bar{F})}{(s - \bar{F})} \). Let \( F'=(1,2,\ldots,m) \) be the set of remote file indexes that are chosen by solving the following problem using Algorithm (H):

\[
\begin{align*}
\text{Max} & \quad \Sigma_{i \in I^+} c_i \bar{F}_i \\
\text{subject to} & \quad \Sigma_{i \in I^+} \bar{F}_i \leq s + \Delta s, \quad x_i \in \{0,1\}, \quad \forall i \in I^+.
\end{align*}
\]

Note that \( F' \) satisfies \( b_1 \geq b_2 \geq \ldots \geq b_m \). There exists an index \( j, 1 \leq j \leq m \), which satisfies \( s - \bar{F} \leq \Sigma_{i=1}^j \bar{F}_i \leq s \). Let us partition set \( F' \) into

\( J=(1,2,\ldots,j) \) and \( \bar{J}=(j+1,j+2,\ldots,m) \). Then we have
\[
\Sigma_{i \in \bar{J}} \bar{F}_i = \Sigma_{i \in F'} \bar{F}_i - \Sigma_{i \in J} \bar{F}_i \leq (s - \bar{F}).
\]

Since \( \Sigma_{i \in F'} \bar{F}_i \leq s + \Delta s \), we obtain
\[
\Sigma_{i \in \bar{J}} \bar{F}_i \leq \Delta s + \bar{F}.
\]

It can be easily shown that if \( b_1 \geq b_2 \geq \ldots \geq b_m \), then
\[
\Sigma_{i \in J} c_i \geq \Sigma_{i \in \bar{J}} \bar{F}_i / \Sigma_{i \in \bar{J}} \bar{F}_i \times \Sigma_{i \in J} c_i.
\]

Therefore,
\[
V(s) \geq \Sigma_{i \in J} c_i \geq \frac{(s - \bar{F})/(\Delta s + \bar{F})}{\Sigma_{i \in \bar{J}} c_i} \times \Sigma_{i \in J} c_i.
\]

Let \( \beta = (s - \bar{F})/(\Delta s + \bar{F}) \). Then
\[
\frac{[V(s+\Delta s)-V(s)]}{V(s)} = \frac{\Sigma_{i \in J} c_i \bar{F}_i + \Sigma_{i \in \bar{J}} c_i}{V(s)} \leq \frac{[V(s)+(V(s)/\beta)]}{V(s) - 1} - 1/\beta = \frac{(\Delta s + \bar{F})}{(s - \bar{F})}. \quad □
\]

The last term \( (\Delta s + \bar{F})/(s - \bar{F}) \) in Inequality (4.13) becomes approximately \( \Delta s/s \), if \( \Delta s \gg \bar{F} \). Furthermore, this condition \( \Delta s \gg \bar{F} \) is realistic since the storage space can be incremented usually by unit of
a disk pack. Therefore, the effect of storage expansion is that the percentage increase in the savings in total traffic load can at most be identical to the percentage increase in the storage capacity.
Users file access behavior in local area environment is highly irregular and hard to predict. But empirical studies have found that there exits time locality of references in this environment. Based on such observation, we developed an adaptive file migration policy which determines the optimal set of remote files to be stored at a local site for a short time future period. The migration decision is based on the Bayesian estimate of the probability of each remote file to be referenced during the short time period. It has been analytically shown that multi-period independent decisions on file migration can successfully minimize the traffic load on the network over a long term horizon, if there is contention among remotely referenced files for the local storage space; viz., if there is strong demand for remote file access.

The time locality of references in file access pattern suggests that a demand policy which unconditionally transfers a remote file copy to a requesting site may perform well if the optimal timing of deleting file copies from the local site can be found. Trace-driven simulations can be used against real data to compare the performance of the heuristic presented in this Part with different migration algorithms.
Appendix A. Proof of Theorem 1

Since we employ a B+ tree-structured primary key directory with each node stored as a block, the number of block accesses $q_1$ to the primary directory in order to locate the address of a desired page would be $\left\lceil \log \frac{1b'}{(1a' + 1_k)} \right\rceil \left\lceil \frac{N}{b^s} \right\rceil$, where $1a'$ is the length in bytes of the address pointer in the primary key directory, $1b'$ is the length in bytes of a block of index entries in the primary key directory, $1_k$ is the length in bytes of the primary key attribute, and $b^s$ is blocking factor of the subfile that contains the primary key attribute. Under Assumption 6, $q_1$ can be restricted to a constant by appropriately choosing the value of $1b'$, so that $q_1$ is not affected by the record segmentation decision.

The processing of an atomic query via the inverted directory takes the following steps [19]: (i) access block index block, (ii) search block index block, (iii) access key-value index block(s), (iv) search key-value index block, (v) access address list block(s), (vi) search address list block(s), (vii) access subfile(s). For a single key disjunctive query, a step is added between steps (vi) and (vii) to merge the address lists for each atomic predicates. Disregarding the CPU time, only steps (i), (i'ii), (v) and (vii) incur accesses to the
external memory and need time considerations. This is also true for a multiple key query. The minimum number of block accesses $q_2$ to the inverted directory (steps (i), (iii) and (v)) in order to process a type 2 query $Q_j$ would be $2 + \left\lceil \frac{r_j}{lb''/la''} \right\rceil$, where $la''$ is the length in bytes of a TID, and $lb''$ is the length in bytes of a block of TID's in the address list area. Clearly, $q_2$ is not affected by the record segmentation decision. □
Appendix B. Proof of Corollary 1

Physical addresses of the response set of multiple key queries, whatever form the selection formula may take, can be found by accessing only the inverted directory, if there is a secondary index for every attribute in the selection formula. By Theorem 1, the cost of accessing the inverted directory does not affect the record segmentation decision. Therefore, multiple key queries of this kind would have the same cost function as operation type 2. □
Appendix C. Proof of Theorem 2

We want to show that \( Z_1^* = Z_{1-\ell}^* \). Given a specific value of \((X_1, X_2)\) that is feasible in \((ILP)_1\), the solution vector \(W\) of \((ILP)_\ell\) is determined so that \( Z_I(\ell)(W) = Z(X_1, X_2) \) due to the equivalence of \((NLP)\) and \((P)\), where the vector \(W\) includes that value of \((X_1, X_2)\). Now, \( Z(X_1, X_2) = Z(X_2, X_1) = Z(\tilde{X}_1, \tilde{X}_2) \) and if \(1R_1 = \ell\), then \(1R_2 = 1-\ell\). Thus, if some \((X_1, X_2)\) that satisfies \(1R_1 = \ell\) and \(1R_2 = 1-\ell\) minimizes \(Z_I(\ell)\), then \((\tilde{X}_1, \tilde{X}_2)\) satisfies \(1R_1 = \frac{\ell}{1-\ell} \sum_{i=1}^{\ell} X_{i1} - \ell\) and \(1R_2 = \frac{\ell}{1-\ell} \sum_{i=1}^{\ell} X_{i2} - \ell\) and also minimizes \(Z_I(1-\ell)\). Therefore, \((X_1, X_2)^*_I(1-\ell) = (\tilde{X}_1, \tilde{X}_2)^*_I(\ell)\), and \(Z_I(\ell) = Z^*_I(1-\ell)\). \(\square\)
Appendix D. Subroutine SUBAL1

Details of the data type declaration and the procedures for implementing the queue facility is omitted. For the reader's convenience for reference, however, the data types and the procedures will be named after those appearing in [3; ch. 2], wherever possible. The main procedure is the same as Algorithm RSEGH except that the following declaration of an abstract data type is required.

type QUEUE = queue with maximum size M;

procedure SUBAL2( var X₁, X₂: X; var Zₜ: real );

var X₁, X₂: X;

Q : QUEUE; (* Q stores fixed attributes as its elements. *)

FIXED : array[1..M] of Boolean; (* FIXED[i] := true, if attribute i is currently fixed; false, otherwise *)

FX : integer; (* most recently fixed attribute *)

FR : integer; (* most recently freed attribute *)

begin

X₁ := X₁; X₂ := X₂;

BACKTRACK := false;

MAKENULL( Q ); (* makes queue Q an empty list *)

INITIAL( FIXED ); (* set all elements of FIXED as false *)
repeat

FX := 0 (* initialize FX as null *)

if BACKTRACK (* if in a backtracking pass *)

then INITIAL(FIXED) (* open all the attributes for test *)

for i := 1 to M do (* for each attribute in the relation R *)

if not FIXED[i] (* considering only the open attributes *)

then begin (* find the attribute that will most reduce the
cost when reassigned to other subfile *)

SWAP(X^1[i], X^2[i]); (* reassign attribute i *)

Z_h := Z(X^1, X^2);(*calculate the objective value of
the changed solution *)

if Z_h < Z_h (* if the objective value is improved *)

then begin

Z_h := Z_h;(*update the incumbent value *)

FX := i (* fix attribute i *)

end

else

SWAP(X^1[i], X^2[i]); (* restore the
assignment of attribute i *)

end (* testing improvements in objective value *)
if FX ≠ 0 (* if any attribute has been fixed improving the objective value *)

then begin

BACKTRACK := false;
ENQUEUE( FX, Q ); (* insert the fixed attribute at the end of queue Q *)
FIXED( FX ) := true; (* close the fixed attribute *)
SWAP( X_1[ FX ], X_2[ FX ] ); (* fix the reassignment *)
X_1^0 := X_1; X_2^0 := X_2; (* save the improved solution *)
end

else begin (* a backtracking pass *)

if BACKTRACK (* if already in backtracking *)

then SWAP( X_1[ FR ], X_2[ FR ] ); (* restore the assignment of most recently freed attribute *)

if EMPTY( Q ) or SIZE_OF( Q ) = 1 (* if queue Q is empty or has only one fixed attribute *)

then begin (* return to main procedure *)

X_1 := X_1^0; X_2 := X_2^0; (* return the best improved solution *)

return

end
else begin (* backtrack to next earliest fixed attribute *)
   BACKTRACK := true;
   DEQUEUE( FR, Q ); (* free the first fixed attribute from queue Q *)
   SWAP( \[ X_1 \], \[ X_2 \] ); (* reassign this freed attribute *)
   \[ Z_h := Z( X_1 X_2 ) ; \] (* calculate the objective value of the changed solution *)
   if \[ Z_h < \hat{Z}_h \] (* if the objective value is improved *)
   then begin (* update the incumbent *)
      \[ \hat{Z}_h := Z_h ; \]
      \[ X_1^O := X_1 ; X_2^O := X_2 \]
   end
end

until FULL( Q ) (* until all the attributes are fixed *)
end; (* SUBALL *)
/* Query Optimization (QOP) */
(defun qop ()
  (prog ()

/* Input */
(print '(Please input p:))
(setq p (read line tty))
(print '(Now please input s:))
(setq s (read line tty))

/* Initiation */
(setq answer nil)
(setq rules nil)
(setq il nil)
(setq ii2 nil)
(setq i2 nil)
(setq i nil)
(setq cind nil)
(setq nrand nil)
(setq both nil)
(setq i nil)
(setq ii nil)
(setq pl (intersection p r1))
(setq p2 (intersection p r2))
(setq sc (intersection s c))
(setq sk (intersection s k))
(setq sn (intersection s n))
(setq snl (intersection sn r1))
(setq sn2 (intersection sn r2))

/* 1 */
(cond ((and (eq (null (union snl pi)) nil) (null (union sn p2)))
  (setq il t) (setq rules (append rules '(1))))

/* 2 */
(cond ((and (eq (null (union sn p2)) nil) (null (union snl pi)))
  (setq i2 t) (setq rules (append rules '(2))))

/* 3 */
(cond ((and (eq (null snl nil))
  (eq (null (union sn2 p2)) nil))
  (setq both t) (setq rules (append rules '(3))))

/* 4 */
(cond ((and (eq (null pl nil)) (eq (null p2 nil)) (null sn))
  (setq i1 t) (setq i2 t) (setq rules (append rules '(4))))

/* 5 */
(cond ((and both (eq (null snl nil)) (null sn2))
  (setq i1 t) (setq ii t) (setq rules (append rules '(5))))

180
/* 6 */
(cond ((and both (eq (null sn2) nil) (null sn1))
        (setq i2 t) (setq iil t) (setq rules (append rules '(6))))
/* 7 */
(cond ((and both (eq (null sn1) nil)
            (eq (null sn2) nil)
            (or (< rho_r1 rho_r2) (eq rho_r1 rho_r2))
            (setq i1 t) (setq iil t) (setq rules (append rules '(7))))
/* 8 */
(cond ((and both (eq (null sn1) nil) (eq (null sn2) nil)
        (greaterp rho_r1 rho_r2)
        (setq i2 t) (setq iil t) (setq rules (append rules '(8))))
/* 9 */
(cond ((and (null sk) (null sn)) (setq nrand t)
        (setq rules (append rules '(9))))
/* 10 */
(cond ((eq (null sc) nil)
        (setq cind t)
        (setq answer (append answer '(scind))
        (setq rules (append rules '(10))))
/* 11 */
(cond ((and il cind (null sk))
        (setq i t) (setq answer (append answer '(seqpl))
        (setq rules (append rules '(11))))
/* 12 */
(cond ((and il (eq cind nil) (null sk))
        (setq i t) (setq answer (append answer '(scanl))
        (setq rules (append rules '(12))))
/* 13 */
(cond ((and il cind (eq (null sk) nil)
        (or (eq (/ r_ck r_c) tau_l) (greaterp (/ r_ck r_c) tau_l))
        (setq i t) (setq answer (append answer '(seqpl))
        (setq rules (append rules '(13))))
/* 14 */
(cond ((and il (eq cind nil) (eq (null sk) nil)
        (or (greaterp (/ r_k no) tau_l) (eq (/ r_k no) tau_l))
        (setq i t) (setq answer (append answer '(seqpl))
        (setq rules (append rules '(14))))
/* 15 */
(cond ((and il cind (eq (null sk) nil) (< (/ r_ck r_c) tau_l))
        (setq i t) (setq answer (append answer '(sind int randl))
        (setq rules (append rules '(15))))
/* 16 */
(cond ((and il (eq cind nil) (eq (null sk) nil) (< (/ r_k no) tau_l))
        (setq i t) (setq answer (append answer '(sind int randl))
        (setq rules (append rules '(16))))
/* 17 */
(cond ((and i iil2 nrand (null sc))
        (setq ii t) (setq answer (append answer '(scan2))
        (setq rules (append rules '(17))))
/* 18 */
(cond ((and i iil2 (eq (null sc) nil))
        (setq ii t) (setq answer (append answer '(linkp2 seqp2))
        (setq rules (append rules '(18))))
*/ 19 */
(cond ((and i ii2 (eq nrand nil))
  (eq (null sc) nil) (< (/ r_1 r_c) tau_2))
  (setq ii t) (setq answer (append answer '((linkf2 rand2)))
  (setq rules (append rules '(19)))))

/* 20 */
(cond ((and i ii2 (eq nrand nil) (null sc) (< (/ r_1 no) tau_2))
  (setq ii t) (setq answer (append answer '((linkf2 rand2)))
  (setq rules (append rules '(20)))))

/* 21 */
(cond ((and i ii2 (eq nrand nil) (eq (null sc) nil)
  (or (greaterp (/ r_1 r_c) tau_2) (eq (/ r_1 r_c) tau_2)))
  (setq ii t) (setq answer (append answer '((linkp2 seqp2)))
  (setq rules (append rules '(21)))))

/* 22 */
(cond ((and i ii2 (eq nrand nil) (null sc)
  (or (greaterp (/ r_1 no) tau_2) (eq (/ r_1 no) tau_2)))
  (setq ii t) (setq answer (append answer '((scan2)))
  (setq rules (append rules '(22)))))

/* 23 */
(cond ((and i ii cind (null sk))
  (setq i t) (setq answer (append answer '(seqp2)))
  (setq rules (append rules '(23)))))

/* 24 */
(cond ((and i ii2 (eq cind nil) (null sk))
  (setq i t) (setq answer (append answer '(scan2)))
  (setq rules (append rules '(24)))))

/* 25 */
(cond ((and ii2 cind (eq (null sk) nil)
  (or (eq (/ r_ck r_c) tau_2) (greaterp (/ r_ck r_c) tau_2)))
  (setq ii t) (setq answer (append answer '(seqp2)))
  (setq rules (append rules '(25)))))

/* 26 */
(cond ((and ii2 cind (eq (null sk) nil)
  (or (eq (/ r_k no) tau_2) (greaterp (/ r_k no) tau_2)))
  (setq ii t) (setq answer (append answer '(seqp2)))
  (setq rules (append rules '(26)))))

/* 27 */
(cond ((and ii ii2 cind (eq (null sk) nil) (< (/ r_ck r_c) tau_2))
  (setq i t) (setq answer (append answer '(sind int rand2)))
  (setq rules (append rules '(27)))))

/* 28 */
(cond ((and ii ii2 cind (eq (null sk) nil) (< (/ r_k no) tau_2))
  (setq i t) (setq answer (append answer '(sind int rand2)))
  (setq rules (append rules '(28)))))

/* 29 */
(cond ((and i ii l nrand (null sc))
  (setq ii t) (setq answer (append answer '(scanl)))
  (setq rules (append rules '(29)))))

/* 30 */
(cond ((and i ii l nrand (eq (null sc) nil))
  (setq ii t) (setq answer (append answer '((linkpl seqp1)))
  (setq rules (append rules '(30)))))


Sample Run 1:

>(setq r '(1 2 3 4 5 6 7 8 9 10))
>(setq rl '(1 3 5 7 9))
>(setq r2 '(2 4 6 8 10))
>(setq c '(1))
>(setq k '(3 4 5 6))
>(setq n '(2 7 8 9 10))
>(setq no 10000)
>(setq r_c 4000)
>(setq r_k 500)
>(setq r_ck 80)
>(setq r_1 100)
>(setq r_2 40)
>(setq tau_1 0.016)
>(setq tau_2 0.006)
>(setq rho_rl 2)
>(setq rho_r 21)

> (qop)
(Please input p:)
1 6 8
(Now please input s:)
1 4 7 10
(____ The Answer Is ____:)
(scind seqp2 linkfl randl)
(____ Rules Applied ____:)
(3 8 10 25 31)
Sample Run 2:

>(setq tau_1 0.005)
>(setq tau_2 0.023)

>(qop)
(Please input p:)
1 6 8
(Now please input s:)
1 4 7 10
(_____ The Answer Is _____:)
(SCIND SIND INT RAND2 LINKP1 SEQP1)
(_____ Rules Applied _____:)
(3 8 10 27 33)

Sample Run 3:

>(setq r_c 1)
>(setq r_k 200)
>(setq r_ck 200)
>(setq r_1 200)
>(setq r_2 20)
>(setq tau_1 0.024)
>(setq tau_2 0.011)
>(setq rho_r1 2)
>(setq rho_r2 1)

>(qop)
(Please input p:)
4 7 9
(Now please input s:)
5 8
(_____ The Answer Is _____:)
(scan2 linkfl randl)
(_____ Rules Applied _____:)
(3 6 26 32)

>(quit)
Footnotes

1. If an atomic predicate with the ≠ operator has the primary key as its selection attribute, it will be processed sequentially. If it has a secondary key or a non-key as its selection attribute, it can be replaced with two atomic predicates with > and < operators. This replacement does not change the operating cost of processing the query.

2. The cost model is trivially appended with a storage cost term if we consider different devices for two subfiles. However, if we consider this, dimensions of both time and space must be converted into value terms. This conversion involves the subjective evaluation of the relative importance of time versus space consumptions of hardware devices.

3. Disregarding the CPU time, the cost model is applicable to anticipatory buffering as well as single buffering for sequential processing, because there will be only a trivial difference in I/O time between the two schemes.

4. ⌈a⌉ denotes a ceiling operator, i.e., the greatest integer smaller than or equal to the real number a.
5. \[ \lfloor a \rfloor \] denotes a floor operator, i.e., the smallest integer greater than or equal to the real number \( a \).

6. For example, it would be better to use Eq. (1.4) than Eq. (1.3) for the estimation of the expected number of cylinder accesses. It may be desirable to use different formulae for different queries if that can reduce the computational error without considerably increasing the computation time.

7. Here we are assuming, without loss of generality, that each attribute consists of an integral number of bytes; otherwise, we can change the unit of attribute length accordingly. We can choose the unit to be the greatest common divisor of all the attribute lengths.

8. NP-completeness of Problem (ILP) can be easily proved by showing that it contains a known NP-complete problem as a special case. It is shown in [55] that a 0-1 linear program, where all objective coefficients, constraint coefficients, and right-hand sides are integers, is NP-complete. Problem (ILP) can be restricted to this problem if we measure the hardware parameters such as \( T_a \), \( T_m \), \( T_l \) and \( T_t \) as integers (by using, for example, microsecond as the unit of measurement).

9. The LP-based branch-and-bound enumeration has often been reported to find an optimal solution very early in the search, with the remaining time spent in verifying optimality [56]. This was also observed in our computational study. When the optimal objective value of a subproblem
was greater than \( \hat{Z} \), the subproblem was typically fathomed by the upper bound within a few iterations. For example, the computation time for a 40-attribute problem was reduced from 1 hour to a couple of seconds when the LP-based method was used with \( \hat{Z} \) as the upper bound.

10. For such cases, the solution time of the (LP) was reduced from 10 to less than 1 second, when a 10-attribute problem was solved and from 35 to 2 seconds for a 30-attribute problem. Even in cases when \( \lambda \) was greater than \( \hat{\lambda} \) by as much as 30, the solution time of (LP)\(_{\lambda} \) was reduced by more than two thirds.

11. The run time of Nauss' knapsack algorithm for solving the (LR) took less than one second for a 40-attribute problem, which was even faster than the (LP) solution method described in the previous section.

12. Prime 9955 is a super mini-computer whose speed relative to IBM 370/195 mainframes is about 1/6.

13. This total run time includes all the overheads such as setting up each subproblem and its relaxation problems in main memory, invoking the Sciconic/VM package and converting the input matrix of each subproblem into a problem file that is executable by the package, passing values or messages among different sub-algorithms by calling the operating system, and so forth. The CPU time spent for all of these overheads usually far exceeds the sum of the run times of all sub-algorithms that make up Algorithm RSEG.
14. Readers can refer to Fig. 1 in [137] to observe the gap between two measures. This gap is largest at the point where the number of tuples retrieved is equal to the total number of pages in a table.

15. It is reported in [11] that, out of 900 experiments, the heuristic ranking failed to produce the optimal sequence in only 9 cases. This shows the effectiveness of using the worst case measure in lieu of the expected measure. Hammer and Niamir [64] employed an even simpler heuristic that first accesses a subfile with the smallest number of blocks, and found that this heuristic was still within 11% of the optimal sequencing.
LIST OF REFERENCES


