SEARCHING MULTIMEDIA DATABASES BY CONTENT

SEARCHING MULTIMEDIA DATABASES BY CONTENT

Christos FALOUTSOS
University of Maryland
College Park, MD, USA

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

Dedication

To my wife Christina and my parents Sophia and Nikos.

CONTENTS

PREFACE

The problem on target is the searching of large multimedia databases by content. For example, '*given a collection of color images, find the ones that look like a sunset*'. Research on a specific domain (eg., machine vision, voice processing, text retrieval) typically focuses on feature extraction and similarity functions, with little regard to the efficiency of the search. Conversely, database research has focused on fast searching for a set of numbers or strings or vectors.

The main goal of this book is to try to bridge the gap between the database and signal processing communities. The book provides enough background information for both areas, presenting the intuition and the mechanics of the best tools in each area, as well as discussing when and how these tools can work together.

The structure of the book reflects its goal. The first half of the book reviews the most successful database access methods, in increasing complexity. It starts from primary-key access methods, where B-trees and hashing are the industry work-horses, and continues with methods that handle n-dimensional vectors. A chapter is also devoted to text retrieval, because text is important on its own right, and because it has led to some extremely useful ideas, like relevance feedback, clustering and the vector-space model. In all the sections, the emphasis is on practical approaches that have been incorporated in commercial systems, or that seem very promising.

The second half of the book uses the above access methods to achieve fast searching in a database of signals. In all cases, the underlying idea is to extract n features from each signal (eg, the first n Discrete Fourier Transform (DFT) coefficients), to map a signal into a point in n-dimensional space; subsequently, the access methods of the first part can be applied to search for similar signals in time that is much faster than sequential scanning, *without* missing any signals that sequential scanning would find ('complete' searching). Then, the book presents some recent, successful applications of this approach on time series and color images. It also describes methods to extract automatically features

from a distance function, using the so-called Multidimensional Scaling (MDS), as well as a newer, faster approximation, called 'FastMap'.

Finally, the appendix gives some background information on fundamental signal processing and linear algebra techniques: the traditional Discrete Fourier Transform (DFT), the Discrete Cosine Transform (used in the JPEG standard), the Discrete Wavelet transform, which is the state-of-the-art in signal processing, the Karhunen-Loeve transform for optimal dimensionality reduction, and the closely related Singular Value Decomposition (SVD), which is a powerful tool for approximation problems. In all the above discussions, the emphasis is on the physical intuition behind each technique, as opposed to the mathematical properties. Source code is also provided for several of them.

The book is targeted towards researchers and developers of multimedia systems. It can also serve as a textbook for a one-semester graduate course on multimedia searching, covering both access methods as well as the basics of signal processing. The reader is expected to have an undergraduate degree in engineering or computer science, and experience with some high-level programming language (eg., 'C'). The exercises at the end of each chapter are rated according to their difficulty. The rating follows a logarithmic scheme similar to the one by Knuth [Knu73]:

00 Trivial - it should take a few seconds

10 Easy - it should take a few minutes

20 It should take a few hours. Suitable for homework exercises.

30 It should take a few days. Suitable for a week-long class project.

40 It should take weeks. Suitable for a semester class project.

50 Open research question.

Acknowledgements Several friends and colleagues have helped in this effort. In alphabetic order: Rakesh Agrawal, Howard Elman, Will Equitz, Myron Flickner, H.V. Jagadish, Philip (Flip) Korn, King-Ip (David) Lin, Yannis Manolopoulos, Wayne Niblack, Douglas Oard, Dragutin Petkovic, M. Ranganathan, Arun Swami, and Kuansan Wang. The research funding of the National Science Foundation (NSF) is also gratefully acknowledged (IRI-8958546, IRI-9205273).

1

INTRODUCTION

As a working definition of a Multimedia Database System we shall consider a system that can store and retrieve multimedia objects, such as 2-dimensional color images, gray-scale medical images in 2-d or 3-d (eg., MRI brain scans), 1-dimensional time series, digitized voice or music, video clips, traditional data types, like 'product-id', 'date', 'title', and any other user-defined data types. For such a system, what this book focuses on is the design of fast searching methods by content. A typical query by content would be, eg., '*in a collection of color photographs, find ones with a same color distribution like a sunset photograph*'.

Specific applications include the following:

- Image databases, where we would like to support queries on color, shape and texture [NBE+93].
- Financial, marketing and production time series, such as stock prices, sales numbers etc. In such databases, typical queries would be '*find companies whose stock prices move similarly*', or '*find other companies that have similar sales patterns with our company*', or '*find cases in the past that resemble last month's sales pattern of our product*'
- Scientific databases, with collections of sensor data. In this case, the objects are time series, or, more general, *vector fields*, that is, tuples of the form, eg., $< x, y, z, t,$ *pressure*, *temperature*, $\ldots >$. For example, in weather data [CoPES92], geological, environmental, astrophysics [Vas93] databases, etc., we want to ask queries of the form, e.g., '*find past days in which the solar magnetic wind showed patterns similar to today's pattern*' to help in predictions of the earth's magnetic field [Vas93].

- multimedia databases, with audio (voice, music), video etc. [NC91]. Users might want to retrieve, eg., similar music scores, or video clips.
- Medical databases, where 1-d objects (eg., ECGs), 2-d images (eg., X-rays) [PF94] and 3-d images (eg., MRI brain scans) [ACF+93] are stored. Ability to retrieve quickly past cases with similar symptoms would be valuable for diagnosis, as well as for medical teaching and research purposes.
- text and photograph archives [Nof86], digital libraries [TSW+85] [Har94] with ASCII text, bitmaps, gray-scale and color images.
- office automation [MRT91], electronic encyclopedias [ST84] [GT87], electronic books [YMD85].
- DNA databases [AGM+90] [WZ96] where there is a large collection of long strings (hundred or thousand characters long) from a four-letter alphabet (A,G,C,T); a new string has to be matched against the old strings, to find the best candidates. The distance function is the editing distance (smallest number of insertions, deletions and substitutions that are needed to transform the first string to the second).

It is instructive to classify the queries of interest in increasing complexity. Consider, for illustration, a set of employee records, where each record contains the employee number `emp#`, `name`, `salary`, `job-title`, a resume (ASCII text), a greeting (digitized voice clip) and a photograph (2-d color image). Then, the queries of interest form the following classes.

primary key '*Find the employee record with* `emp#`*= 123*'. That is, the specified attribute has no duplicates.

secondary key '*Find the employee records with* `salary`*=40K and* `job-title` *= engineer*'. That is, the queries involve attributes that may have duplicate values.

text '*Find the employee records containing the words* 'manager', 'marketing' *in their resume*'. A text attribute contains an unspecified number of alphanumeric strings.

signals For example, a query on 1-d signals could be '*Find the employee records whose greeting sounds similar to mine*'. Similarly, for 2-d signals, a query could be '*Find employee photos that look like a desirable photo*'.

Acronym	Definition
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DNA	DeoxyriboNucleic Acid
DWT	Discrete Wavelet Transform
GEMINI	GEneric Multimedia INdexIng method
GIS	Geographic Information Systems
IR	Information Retrieval
LSI	Latent Semantic Indexing
MBR	Minimum Bounding Rectangle
MDS	Multi-Dimensional Scaling
MRI	Magnetic Resonance Imaging
PAM	Point Access Method
SAM	Spatial Access Method
SVD	Singular Value Decomposition
SWFT	Short-Window Fourier Transform
WWW	World-Wide-Web

Table 1.1 Summary of Acronyms and Definitions

The book is organized in two parts and an appendix, following the above classification of queries. In the first part, we present the most successful methods for indexing traditional data (primary, secondary, and text data). Most of these methods, like B-trees and hashing, are textbook methods and have been successfully incorporated in commercial products. In the second part we examine methods for indexing signals. The goal is to adapt the previously mentioned tools and to make them work in this new environments. Finally, in the appendix we present some fundamental techniques from signal processing and matrix algebra, such as the Fourier transform and the Singular Value Decomposition (SVD).

Table 1.1 gives a list of the acronyms that we shall use in the book.

PART I

DATABASE INDEXING METHODS

2

INTRODUCTION TO RELATIONAL DBMS

This chapter presents the minimal necessary set of concepts from relational database management systems. Excellent textbooks include, eg., [KS91] [Dat86]. The chapter also describes how the proposed methods will fit in an extensible DBMS, customized to support multimedia datatypes.

Traditional, relational database management systems ('RDBMS' or just 'DBMS') are extremely popular. They use the *relational model* [Cod70] to describe the data of interest. In the relational model, the information is organized in *tables* ('relations'); the rows of the tables correspond to records, while the columns correspond to attributes. The language to store and retrieve information from such tables is the *Structured Query Language* (SQL).

For example, if we want to create a table with employee records, so that we can store their employee number, name, age and salary, we can use the following SQL statement:

```
create table EMPLOYEE (
    emp# integer,
    name char(50),
    age float,
    salary float);
```

The result of the above statement is to notify the DBMS about the EMPLOYEE table (see Figure 2.1). The DBMS will create a table, which will be empty, but ready to hold EMPLOYEE records.

Tables can be populated with the SQL `insert` command. E.g.

EMPLOYEE	emp#	name	age	salary

Figure 2.1 An empty EMPLOYEE table ('relation').

```
insert into EMPLOYEE values (
   123, "Smith, John", 30, 38000.00);
```

will insert a row in the EMPLOYEE table, recording the information about the employee 'Smith'. Similarly, the command to insert the record for another employee, say, 'Johnson', is:

```
insert into EMPLOYEE values (
   456, "Johnson, Tom", 25, 55000.00);
```

The result is shown in Figure 2.2

EMPLOYEE	emp#	name	age	salary
	123	Smith, John	30	38000.00
	456	Johnson, Tom	25	55000.00

Figure 2.2 The EMPLOYEE table, after two `insertions`

We can retrieve information using the `select` command. E.g., if we want to find all the employees with salary less than 50,000, we issue the following query:

```
select *
from EMPLOYEE
where salary <= 50000.00
```

In the absence of indices, the DBMS will perform a sequential scanning, checking the salary of each and every employee record against the desired threshold of 50,000. To accelerate queries, we can create an index (usually, a B-tree index, as described in Chapters 3 and 4), with the command `create index`. For

example, to build an index on the employee's salary, we would issue the SQL statement:

```
create index salIndex on EMPLOYEE (salary);
```

SQL provides a large number of additional, valuable features, such as the ability to retrieve information from several tables ('joins') and the ability to perform aggregate operations (sums, averages). However, we restrict the discussion to the above few features of SQL, which are the absolutely essential ones for this book.

Every commercial DBMS offers the above functionalities, supporting numerical and string datatypes. Additional, user-defined datatypes, like images, voice etc., need an *extensible DBMS*. Such a system offers the facility to provide new data types, along with functions that operate on them. For example, one datatype could be 'voiceClip', which would store audio files in some specified format; another datatype could be 'image', which would store, eg., JPEG color images. The definition of new datatypes and the associated functions for them ('display', 'compare' etc.) are typically implemented by a specialist. After such datatypes have been defined, we could create tables that can hold multimedia employee records, with the command, eg.:

```
create table EMPLOYEE (
    emp# fixed,
    name char(50),
    salary float,
    age float,
    greeting voiceClip,
    face image);
```

Assuming that the predicate `similar` has been appropriately defined for the 'image' datatype, we can look for employees that look like given person, as follows:

```
select name
from EMPLOYEE
where EMPLOYEE.face similar desirableFace
```

where 'desirableFace' is the object-id of the desirable JPEG image.

Providing the ability to answer such queries is exactly the focus of this book. The challenges are two: (a) how to measure 'similarity' and (b) how to search efficiently. In this part of the book we examine older database access methods that can help accelerate the search. In the second part we discuss some similarity measures for multimedia data types, like time sequences and color images.

Before we continue with the discussion of database access methods, we should notice that they are mainly geared towards a two-level storage hierarchy:

- The first level is fast, small, and expensive. Typically, it is the *main memory* or *core* or *RAM*, with an access time of micro-seconds or faster.
- The second level (*secondary store*) is much slower, but much larger and cheaper. Typically, it is a magnetic disk, with ≈5-10 msec access time.

Typically, database research has focused on large databases, which do not fit in main memory and thus have to be store on secondary store. A major characteristic of the secondary store is that it is organized into *blocks* (= *pages*). The reason is that, accessing data from the disk involves the mechanical move of the read/write head of the disk above the appropriate track on the disk. Exactly because these moves ('seeks') are slow and expensive, every time we do a disk-read we bring into main memory a whole disk *block*, of the order of 1Kb-8Kb. Thus, it makes a huge performance difference if we manage to group similar data in the same disk blocks. Successful access methods (like the B-trees) try exactly to achieve good clustering, to minimize the number of disk-reads.

3

PRIMARY KEY ACCESS METHODS

Here we give a brief overview of the traditional methods to handle queries on primary (ie, unique) keys. Considering the running example of EMPLOYEE records, a typical query is, eg., '*find the employee with* `emp#` *= 344*'. Notice that the `emp#` is assumed to have no duplicates.

Primary key access methods are useful for multimedia data for two reasons:

1. primary keys will be part of the information: for example, in an employee database, we may have the `emp#` as the primary key; in a video database, we may have the title or the ISBN as the primary key, etc.
2. The primary key access methods provide fundamental ideas, like the hierarchical organization of records that the B-trees suggest. These ideas were the basis for newer, more general and more powerful access methods, like the R-trees (see Section 5.2), that can be used for multimedia data as well.

For primary keys on secondary store, the textbook approaches are two: B-trees and hashing [Knu73].

3.1 HASHING

The idea behind hashing is the *key-to-address transformation*. For example, consider a set of 40,000 employee records, with unique 9-digit `emp#`. Also assume that we are interested in providing fast responses for queries on `emp#`.

Suppose we have decided to use 1,000 consecutive disk pages (=blocks = buckets), each capable of holding 50 records. Then, we can use a *hashing function* $h()$, to map each key to a bucket. For example:

$$h(emp\#) \;=\; (emp\#) \mod 1000 \tag{3.1}$$

is a function that maps each `emp#` to its last three digits, and therefore, to the corresponding bucket, as shown in Figure 3.1

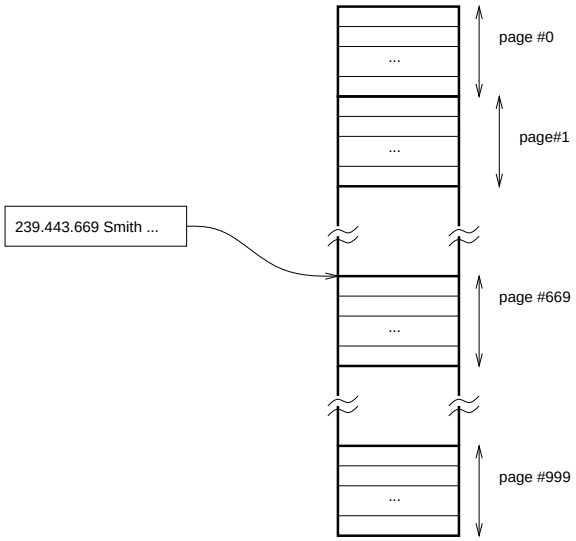


Figure 3.1 Illustration of a hash table, with 'division hashing' ($h(emp\#) = (emp\#) \mod 1000$).

The first step in the design of a hashing scheme is the choice of the hashing function. There are several classes of them, the most successful ones being (a) the *division hashing*, like the function of Eq. 3.1 and (b) the *multiplication hashing*.

The second step in the design of a hashing scheme is to choose a collision resolution method. Notice that we deliberately allocate more space in the hash table than needed: in our example, 50,000 slots, versus 40,000 records; in general, we opt for 80%-90% load factor. However, due to the almost random nature of the hashing function, there is always the possibility for bucket-overflows. In such a case, we have several choices, the most popular being: (a) using a separate overflow area (*'separate chaining'*) and (b) re-hashing to another bucket (*'open addressing'*).

There are numerous surveys, variations and analyses of hashing [Kno75, SD76, Sta80, Lar85]. An easily accessible hash-table implementation is the `ndbm` package of UNIX^{TM}, which uses hashing with 'open addressing'.

3.1.1 Extensible hashing

The original hashing suffered from the fact that the hash table can not grow or shrink, to adapt to the volume of insertions and deletions. The reason is that the size of the table is 'hardwired' in the hashing function; changing the size implies changing the hashing function, which may force relocation of each and every record, a very expensive operation.

Relatively recent developments tried to alleviate this problem by allowing the hash table to grow and shrink without expensive reorganizations. These methods come under the name of extensible hashing: extendible hashing [FNPS79], dynamic hashing [Lar78], spiral hashing [Mar79], linear hashing [Lit80], linear hashing with partial expansions [Lar82]. See [Lar88] for a recent survey and analysis of such methods.

3.2 B-TREES

B-trees and variants are among the most popular methods for physical file organization on disks [BM72]. Following Knuth [Knu73], we have:

Definition 3.1 *A B-tree of order m is a multiway tree, with the key-ordering property, satisfying the following restrictions:*

1. *Every node has* $\leq m$ *sons.*
2. *Every node, except for the root, has* $\geq m/2$ *sons.*
3. *The root has at least 2 sons, unless it is a leaf.*
4. *All leaves appear at the same level*
5. *A non-leaf node with k sons contains k-1 keys.*

The key-ordering property means that, for every sub-tree, the root of the sub-tree is greater than all the key values at the left and smaller than all the key values at the right sub-tree. Notice that the leaves are empty; in practice, the leaves and the pointers pointing to them are omitted, to save space.

To achieve a high fan-out, the tree does not store the full records; instead, it stores pointers to the actual records. More specifically, the format of a B-tree node of order m is as follows:

$$(p_1, key_1, ptr_1, p_2, key_2, ptr_2, \ldots, p_m)$$

where ptr_i is the pointer to the record that corresponds to key_i; p_i is a pointer to another B-tree node (or null). Figure 3.2 shows a B-tree of order m=3 and height 2. Notice that (a) the pointers to the records are not shown (b) the tree fulfills the B-tree properties.

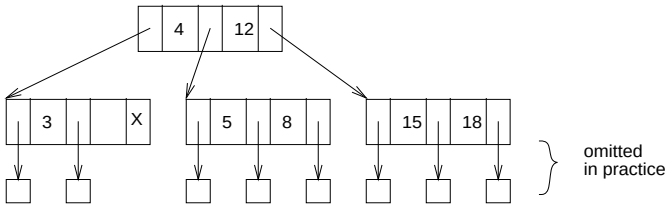


Figure 3.2 Illustration of a B-tree of order m=3. 'X' indicates null pointers.

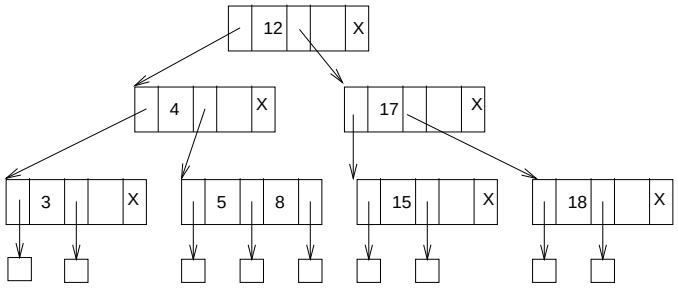


Figure 3.3 Insertion of key #17 in the previous B-tree.

Because of the above definition, a B-tree has the following desirable characteristics:

- it is always balanced, thus leading to logarithmic search time $O(\log_m(N+1))$ and few disk accesses.

- it has guaranteed 50% space utilization, while the average is $\approx 69\%$ [Yao78, LW89, JS89]
- it also has logarithmic insertion and deletion times.

The insertion and deletion algorithms are masterfully designed [BM72] to maintain the B-tree properties. A full discussion is outside the scope of this book (see, eg., [Knu73]). A brief sketch of them is instructive, though:

The insertion algorithm works as follows: given a key to be inserted, we find the appropriate leaf node; if there is not enough space there, then we split the node in two, pushing the middle key to the parent node; the parent node may recursively overflow and split again. Figure 3.3 shows the resulting B-tree, after key '17' is inserted into the B-tree of Figure 3.2. Notice the propagated split, which created a new root, thus increasing the height of the B-tree. Thus, the B-tree 'grows from the leaves'. Notice that a node has the lowest possible utilization (50%) immediately after a split, exactly because we split a 100%-full node into 2 new ones.

Deletion is done in the reverse way: omitting several details, if a node underflows, it either borrows keys from one of its siblings, or it merges with a sibling into a new node.

B-trees are very suitable for disks: each node of the B-tree is stored in one page; typically, the fanout is large, and thus the B-tree has few levels, requiring few disk ($\equiv$node) accesses for a search.

This concludes the quick introduction to the basic B-trees. There are two very successful variations, namely the B^+-trees and the B^*-trees:

- The B^+-trees keep a copy of all the keys at the leaves, and string the leaves together with pointers. Thus the scanning of the records in sorted order is accelerated: after we locate the first leaf node, we just follow the pointer to the next leaf.
- The B^*-trees introduce the idea of *deferred splitting*. Splits hurt the performance, because they create new, half-empty nodes, and potentially they can make the tree taller (and therefore slower). The goal is to try to postpone splits: instead of splitting in two an overflowing node, we check to see if there is a sibling node that could host some of the overflowing keys. If the sibling node is also full, *only then* we do a split. The split though

involves *both* of the full nodes, whose entries are divided among *three* new nodes. This clever idea results in much fewer splits and in guaranteed 66% (= 2/3) space utilization of the nodes, with a higher average than that. These splits are called '2-to-3' splits; obviously, we can have '3-to-4' and 's-to-$(s+1)$' splits. However, the programming complexity and the additional disk accesses on insertion time reach a point of diminishing returns. Thus, the '2-to-3' split policy usually provides a good trade-off between search time and insertion effort.

3.3 CONCLUSIONS

B-trees and hashing are the industry work-horses. Each commercial system provides at least one of them. Such an index is built, eg., by the `create index` command of SQL, as discussed in Chapter 2. The two methods compare as follows:

- B-trees guarantee logarithmic performance for any operation (insertion, deletion, search), while hashing gives constant search on the *average* (with linear performance, in the worst case). Depending on the specific version, the insertion and update times for hashing can be constant, or grow linearly with the relation size.
- B-trees can expand and shrink gracefully, as the relation sizes grows or shrinks; hashing requires expensive reorganization unless an extensible hashing method is used (such as the 'linear hashing').
- B-trees preserve the key order, which allows them to answer range queries, nearest neighbor queries, as well as to support ordered sequential scanning. (Eg., consider the query '*print all the employees' paychecks, in increasing* `emp#` *order*').

Exercises

Exercise 3.1 [15] *In the B-tree of Figure 3.2, insert the key '7'.*

Exercise 3.2 [15] *In the B-tree of Figure 3.3, delete the key '15'.*

Exercise 3.3 [32] *Design and implement the algorithm for insertion and deletion in B-trees.*

Exercise 3.4 [34] *Design and implement the algorithm for insertion and deletion in B^*-trees (i.e., with deferred splitting).*

Exercise 3.5 [25] *Using an existing B-tree package, analyze its average-case space utilization through simulation.*

Exercise 3.6 [25] *Implement a phone-book database, using the* `ndbm` *library of* UNIX™. *Treat the phone number as the primary key.*

4

SECONDARY KEY ACCESS METHODS

Access methods for secondary key retrieval have attracted much interest. The problem is stated as follows: Given a file, say, `EMPLOYEE( name, salary, age)`, organize the appropriate indices so that we can answer efficiently queries on any and all of the available attributes. Rivest [Riv76] classified the possible queries into the following classes, in increasing order of complexity:

- *exact match* query, when the query specifies all the attribute values of the desired record, e.g.:

  `name` = 'Smith' `and salary` = 40,000 `and age` = 45

- *partial match* query, when only some of the attribute values are specified, e.g.:

  `salary` = 40,000 `and age` = 35

- *range queries*, when ranges for some or all of the attributes are specified, e.g.:

  35,000 $\leq$ `salary` $\leq$ 45,000 `and age` = 45

- *Boolean queries*:

  ((`not name` = 'Smith') `and salary` $\geq$ 40,000) `or age` $\geq$ 50

In the above classification, each class is a special case of the next class. A class of queries outside the above hierarchy is the *nearest neighbor* query:

- *nearest neighbor* or *best match* query, eg.:

$$\texttt{salary} \approx 45{,}000 \texttt{ and age} \approx 55$$

where the user specifies some of the attribute values, and asks for the best match(es), according to some pre-specified distance/dis-similarity function.

In this chapter, first we mention the inverted files, which is the industry work-horse. Then we describe some methods that treat records as points in k-d space (where k is the number of attributes); these methods are known as *point access methods* or PAMs, and are closely related to the upcoming *spatial access methods* (SAMs).

4.1 INVERTED FILES

This is the most popular approach in database systems. An inverted file on a given attribute (say, '`salary`') is built as follows: For each distinct attribute value, we store:

1. A list of pointers to records that have this attribute value (*postings list*).
2. Optionally, the length of this list.

The set of distinct attribute values is typically organized as a B-tree or as a hash table. The postings lists may be stored at the leaves, or in a separate area on the disk. Figure 4.1 shows an index on the `salary` of an EMPLOYEE table. A list of unique salary values is maintained, along with the 'postings' lists.

Given indices on the query attributes, complex boolean queries can be resolved by manipulating the lists of record-pointers, before accessing the actual records.

A interesting variation that can handle conjunctive queries has been proposed by Lum [Lum70], by using *combined indices*: We can build an index on the *concatenation* of two or more attributes, for example (`salary`, `age`). Such an index can answer easily queries of the form '`salary`=*40000* `and age`=*30*', without the need of merging any lists. Such an index will contain all the unique, existing pairs of (`salary`, `age`) values, sorted on lexicographical order. For each pair, it will have a list of pointers to the EMPLOYEE records with the specified combination of `salary` and `age`.

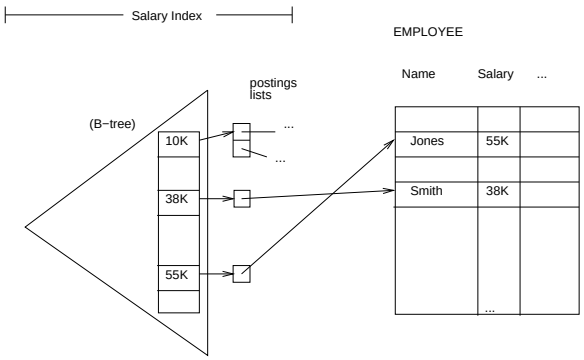


Figure 4.1 Illustration of inversion: a B-tree index on `salary`.

As mentioned in Chapter 2, plain or combined indices can be created automatically by a relational DBMS, with the SQL command `create index`.

4.2 POINT ACCESS METHODS (PAMS)

A fruitful point of view is to envision a record with k attributes as a point in k-dimensional space. Then, there are several methods that can handle points, the so-called *Point Access Methods* (PAMs) [SK90]. Since most of them can also handle spatial objects (rectangles, polygons, etc.) in addition to points, we postpone their description for the next chapter. Here we briefly describe two of the PAMs, the *grid files*, and the *k-d-trees*. They both are mainly designed for points and they have proposed important ideas that several SAMs have subsequently used.

4.2.1 Grid File

The grid file [NHS84] can be envisioned as the generalization of extendible hashing [FNPS79] in multiple dimensions. The idea is that it imposes a grid on the address space; the grid adapts to the data density, by introducing more divisions on areas of high data density. Each grid cell corresponds to one disk page, although two or more cells may share a page. To simplify the 'record-keeping', the cuts are allowed only on predefined points (1/2, 1/4, 3/4 etc. of each axis) and they cut all the way through, to form a grid. Thus, the grid

file needs only a list of cut-points for every axis, as well as a directory. The directory has one entry for every grid cell, containing a pointer to the disk page that contains the elements of the grid cell.

The grid file has the following desirable properties: it guarantees 2 disk accesses for exact match queries; it is symmetric with respect to the attributes; and it adapts to non-uniform distributions. However, it suffers from two disadvantages: (a) it does not work well if the attribute values are correlated (eg., '`age`' and '`salary`' might be linearly correlated in an EMPLOYEE file) and (b) it might need a large directory, if the dimensionality of the address space is high ('dimensionality curse'). However, for a database with low-dimensionality points and un-correlated attributes, the grid file is a solution to consider.

Several variations have been proposed, trying to avoid these problems: the rotated grid file [HN83] rotates the address space, trying to de-correlate the attributes; the tricell method [FR89a] uses triangular as opposed to rectangular grid cells; the twin grid file [HSW88] uses a second, auxiliary grid file, to store some points, in an attempt to postpone the directory growth of the main grid file.

4.2.2 K-d-trees

This is the only main-memory access method that we shall describe in this book. The exception is due to the fact that k-d-trees propose elegant ideas that have been used subsequently in several access methods for disk-based data. Moreover, extensions of the original k-d-tree method have been proposed [Ben79] to group and store k-d-tree nodes on disk pages, at least for static data.

The k-d-tree [Ben75] divides the address space in disjoint regions, through 'cuts' on alternating dimensions/attributes. Structurally, it is a binary tree, with every node containing (a) a data record (b) a left pointer and (c) a right pointer. At every level of the tree, a different attribute is used as the 'discriminator', typically in a round-robin fashion.

Let n be a node, r be the record in this node, and A be the discriminator for this node. Then, the left subtree of the node n will contain records with smaller A values, while the right subtree will contain records with greater or equal A values. Figure 4.2(a) illustrates the partitioning of the address space by a k-d-tree: the file has 2 attributes (eg., '`age`' and '`salary`'), and it contains the

following records (in insertion order): (30,50), (60,10), (45, 20). Figure 4.2(b) shows the equivalent k-d-tree as a binary tree.

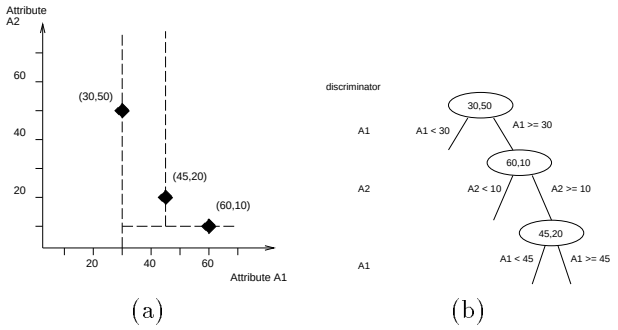


Figure 4.2 Illustration of a k-d tree with three records: (a) the divisions in the address space and (b) the tree itself.

The k-d tree can easily handle exact-match queries, range queries and nearest-neighbor queries [Ben75]. The algorithms are elegant and intuitive, and they typically achieve good response times, thanks to the efficient 'pruning' of the search space that the k-d-tree leads to.

Several disk-based PAMs have been inspired by or used k-d-trees. The k-d-B-trees [Rob81] divide the address space in m regions for every node (as opposed to just 2 that the k-d-tree does), where m is fanout of the tree. The hB-tree [LS90] divides the address space in regions that may have 'holes'; moreover, the contents of every node/disk-page are organized into a k-d-tree.

4.3 CONCLUSIONS

With respect to secondary-key methods, inversion with a B-tree (or hashed) index is automatically provided by commercial DBMS with the `create index` SQL command. The rest of the point access methods are typically used in stand-alone systems. Their extensions, the *spatial access methods*, are examined next.

Exercises

Exercise 4.1 **[33]** *Implement a grid-file package, for n=2 dimensions with insertion and range search routines.*

Exercise 4.2 **[33]** *Modify the previous package, so that the number of dimensions n is user-defined.*

Exercise 4.3 **[30]** *For each of the above packages, implement a 'nearest neighbor' search algorithm.*

Exercise 4.4 **[20]** *Extend your nearest neighbor algorithms to search for k nearest neighbors, where k is user-defined.*

Exercise 4.5 **[30]** *Populate each of the above packages with N=10,000-100,000 points; issue 100 nearest-neighbor queries, and plot the response time of each method, as well as the time for the sequential scanning.*

Exercise 4.6 **[30]** *Using a large database (real or synthetic, such as the Wisconsin benchmark), and any available RDBMS, ask selections queries before and after building an index on the query attributes; time the results.*

5

SPATIAL ACCESS METHODS (SAMS)

In the previous section we examined the so-called 'secondary key' access methods, which handle queries on keys that may have duplicates (eg., '`salary`', or '`age`', in an `EMPLOYEE` file). As mentioned, records with k numerical attributes can be envisioned as k-dimensional points. Here we examine *spatial access methods*, which are designed to handle multidimensional points, lines, rectangles and other geometric bodies.

There are numerous applications that require efficient retrieval of spatial objects:

- Traditional relational databases, where, as we mentioned, records with k-attributes become points in k-d spaces (see Figure 5.1(a)).
- Geographic Information Systems (GIS), which contain, eg., point data, such as cities on a two-dimensional map (see Figure 5.1(b)).
- Medical image databases with, for example, three-dimensional MRI brain scans, require the storage and retrieval of point-sets, such as digitized surfaces of brain structures [ACF+93].
- Multimedia databases, where multi-dimensional objects can be represented as points in feature space [Jag91, FRM94]. For example, 2-d color images correspond to points in (R,G,B) space (where R,G,B are the average amount of red, green and blue [FBF+94]). See Figure 5.1(c).
- Time-sequences analysis and forecasting [WG94, CE92], where k successive values are treated as a point in k-d space; correlations and regularities in this k-d space help in characterizing the dynamical process that generates the time series.

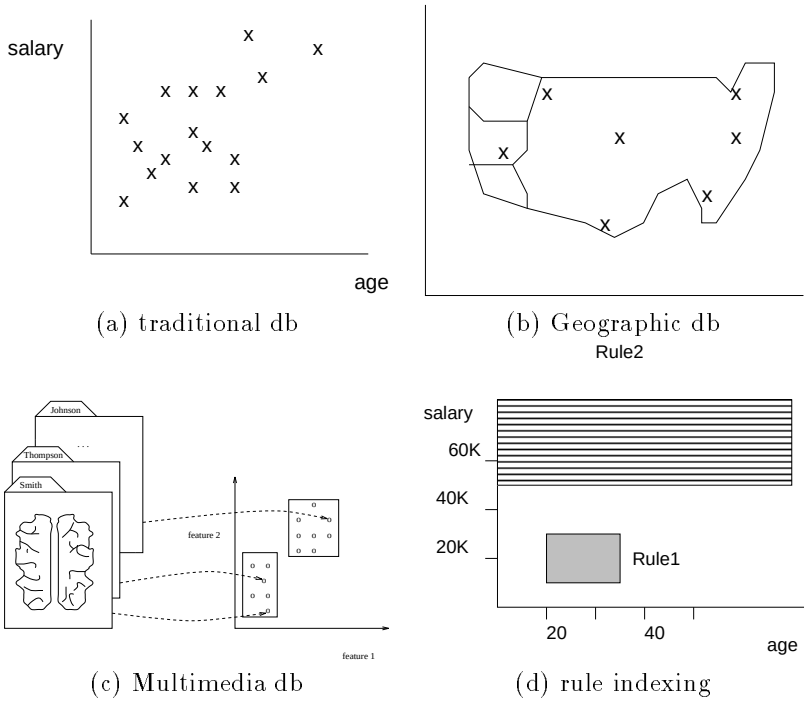
- Rule indexing in expert database systems [SSH86] where rules can be represented as ranges in address space (eg., '*all the employees with salary in the range (10K-20K) and age in the rage (30-50) are entitled to specific health benefits*'). See Figure 5.1(d).

In a collection of spatial objects, there are additional query types that are of interest. The following query types seem to be the most frequent:

1. *range* queries, a slight generalization of the range queries we saw in secondary key retrieval. Eg., '*find all cities within 10 miles from Washington DC*'; or '*find all rivers in Canada*'. Thus the user specifies a region (a circle around Washington, or the region covered by Canada) and asks for all the objects that intersect this region. The *point query* is a special case of the range query, when the query region collapses to a point. Typically, the range query request all the spatial objects that intersect a region; similarly, it could request the spatial objects that *are completely contained*, or that *completely contain* the query region. We mainly focus on the 'intersection' variation; the rest two can usually be easily answered, by slightly modifying the algorithm for the 'intersection' version.
2. *nearest neighbor* queries, again a slight generalization of the nearest neighbor queries for secondary keys. Eg., '*find the 5 closest post-offices to our office building*'. The user specifies a point or a region, and the system has to return the k closest objects. The distance is typically the Euclidean distance (L_2 norm), or some other distance function (eg., city-block distance L_1, or the L_∞ norm etc).
3. *spatial joins*, or *overlays*: eg., in a CAD design, '*find the pairs of elements that are closer than* ϵ' (and thus create electromagnetic interference to each other). Or, given a collection of lakes and a collection of cities, '*find all the cities that are within 10km from a lake*'.

The proposed methods in the literature form the following classes. For a recent, extensive survey, see [GG95].

- Methods that use *space filling curves* (also known as *z-ordering* or *linear quadtrees*).
- Methods that use tree-like structures: R-trees and its variants.



(a) traditional db (b) Geographic db

(c) Multimedia db (d) rule indexing

Figure 5.1 Applications of Spatial Access Methods

The next two sections are dedicated to each of the above classes. For each class we discuss the main idea, its most successful variations, and sketch the algorithms to handle the above query types. In the third section we present the idea that transforms spatial objects into higher-dimensionality points. In the last section we give the conclusions for this chapter.

5.1 SPACE FILLING CURVES

The method has attracted a lot of interest, under the names of N-trees [Whi81], linear quadtrees [Gar82], z-ordering [Ore86] [OM88] [Ore89] [Ore90] etc. The fundamental assumption is that there is a finite precision in the representation of each co-ordinate, say K bits. The terminology is easiest described in 2-d address space; the generalizations to n dimensions should be obvious. Following the quadtree literature, the address space is a square, called an *image* , and it

is represented as a $2^K \times 2^K$ array of 1×1 squares. Each such square is called a *pixel*.

Figure 5.2 gives an example for n=2 dimensional address space, with K=2 bits of precision. Next, we describe how the method handles points and regions.

5.1.1 Handling points

The space filling curve tries to impose a linear ordering on the resulting pixels of the address space, so that to translate the problem into a primary-key access problem.

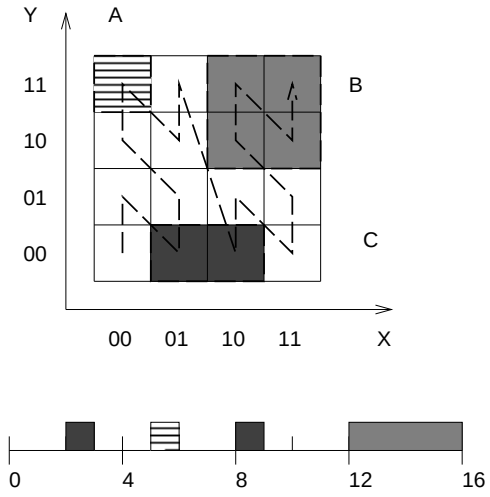


Figure 5.2 Illustration of Z-ordering

One such obvious mapping is to visit the pixels in a row-wise order. A better idea is to use *bit interleaving* [OM84]. Then, the *z-value* of a pixel is the value of the resulting bit string, considered as a binary number. For example, consider the pixel labeled 'A' in Figure 5.2, with coordinates x_A= `00` and y_A= `11`. Suppose that we decide to shuffle the bits, starting from the x-coordinate first, that is, the order with which we pick bits from the coordinates is '1,2,1,2' ('1' corresponds to the x coordinate and '2' to the y coordinate). Then, the z-value z_A of pixel 'A' is computed as follows:

$$z_A = \text{Shuffle}\ (\ \text{'1,2,1,2'}, x_A, y_A) = \text{Shuffle}\ (\text{'1,2,1,2'}, \texttt{00}, \texttt{11}) = \texttt{0101} = (5)_{10}$$

Visiting the pixels in ascending z-value order creates a self-similar trail as depicted in Figure 5.2 with a dashed line; the trail consists of 'N' shapes, organized to form larger 'N' shapes recursively. Rotating the trail by 90 degrees gives 'z' shapes, which is probably the reason that the method was named *z-ordering*. Figure 5.3 shows the trails of the z-ordering for a 2×2, a 4×4 and an 8×8 grid. Notice that each larger grid contains four miniature replicas of the smaller grids, joined in an 'N' shape.

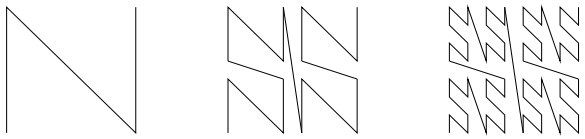


Figure 5.3 Z-order curves for 2x2, 4x4 and 8x8 grids.

We have just described one method to compute the z-value of a point in 2-d address space. The extension to n-d address spaces is obvious: we just shuffle the bits from each of the n dimensions, visiting the dimensions in a round-robin fashion. The inverse is also obvious: given a z-value, we translate it to a binary number and un-shuffle its bits, to derive the n coordinate values.

5.1.2 Handling Regions

The z-value of a region is more complicated. In fact, a region typically breaks into one or more pieces, each of which can be described by a z-value. For example, the region labeled 'C' in Figure 5.2 breaks into two pixels C_1 and C_2, with z-values

$$z_{C_1} = \texttt{0010} = (2)_{10}$$
$$z_{C_2} = \texttt{1000} = (8)_{10}$$

The region labeled 'B' consists of four pixels, which have the common prefix `11` in their z-values; in this case, the z-value of 'B' is exactly this common prefix:

$$z_B = \texttt{11}$$

A conceptually easier and computationally more efficient way to derive the z-values of a region is through the concept of 'quadtree blocks'. Consider the

four equal squares that the image can be decomposed into. Each such square is called a *level-1 block*; a *level-i block* can be recursively defined as one of the four equal squares that constitute a level-$(i-1)$ block. Thus, the pixels are level-K blocks; the image is the (only) level-0 block. Notice that for a level-i block, all its pixels have the same prefix up to $2i$ bits; this common prefix is defined as the z-value of the given block.

We obtain the quadtree decomposition of an object (region) by recursively dividing it into blocks, until the blocks are homogeneous or until we reach the pixel level (level-K blocks). For a 2-dimensional object, the decomposition can be represented as a 4-way tree, as shown in Figure 5.4(b). Blocks that are empty/full/partially-full are represented as white, black and gray nodes in the quadtree, respectively.

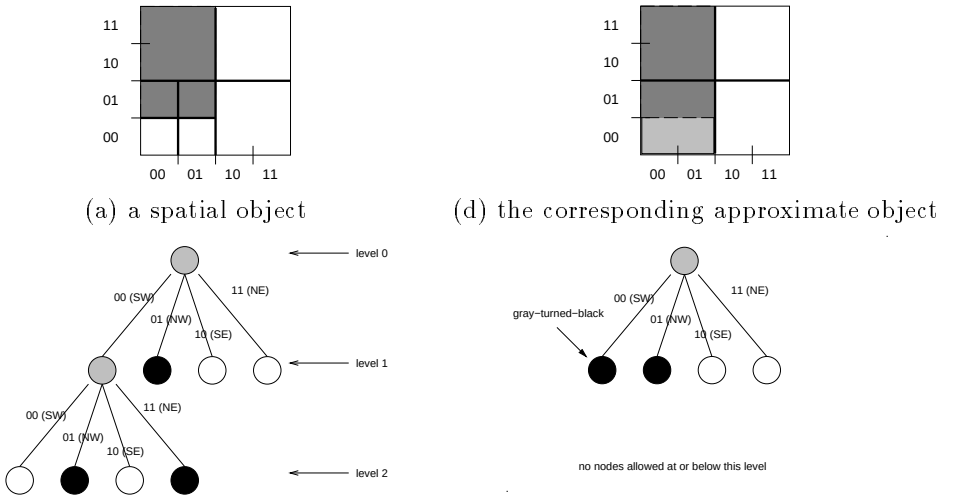
For efficiency reasons (eg., see [Ore89, Ore90]), we typically approximate an object with a 'coarser resolution' object. In this case, we stop the decomposition earlier, eg., when we reach level-i blocks ($i < K$), or when we have a large enough number of pieces. Figure 5.4(c) and (d) give an example.

The quadtree representation gives an easy way to obtain the z-value for a quadtree block: Let '0' stand for 'south' and for 'west', and '1' stand for 'north' and for 'east'. Then, each edge of the quadtree has a unique, 2-bit label, as shown in Figure 5.4; the z-value of a block is defined as the concatenation of the labels of the edges from the root of the quadtree to the node that corresponds to the given block.

Since every block has a unique z-value, we can represent the quadtree decomposition of an object by listing the corresponding z-values. Thus, the z-values of the shaded rectangle in figure 5.4(a) are '0001' (for 'WS WN') '0011' (for 'WS EN') and '01' (for 'WN').

As described above, quadtrees have been used to store objects in main memory. For disk storage, the prevailing approach is the so-called *linear* quadtree [Gar82], or, equivalently the z-ordering method [Ore86]. Each object (and range query) can be uniquely represented by the z-values of its blocks. Each such z-value can be treated as a primary-key of a record of the form (z-value, object-id, *other attributes* ...), and it can be inserted in a primary-key file structure, such as a B^+-tree. Table 5.1 illustrates such a relation, containing the z-values of the shaded rectangle of Figure 5.4(a).

Additional objects in the same address space can be handled in the same way; their z-values will be inserted into the same B^+-tree.



(a) a spatial object (d) the corresponding approximate object

(b) its exact quadtree decomposition (c) an approximate quadtree decomposition

Figure 5.4 Counter-clockwise, from top-left: (a) The shaded rectangle is decomposed into three blocks. (b) the corresponding quadtree, with z-values `01`, `0001` and `0011` (c) an approximate quadtree, with z-values `01`, `00` (d) the corresponding approximate spatial object - the lightly-shaded region is the enlargement, due to the approximation.

z-value	object id	(other attributes)
...	...	...
0001	'ShadedRectangle'	...
...	...	...
0011	'ShadedRectangle'	...
...	...	...
01	'ShadedRectangle'	...
...	...	...

Table 5.1 Illustration of the relational table that will store the z-values of the sample shaded rectangle.

5.1.3 Algorithms

The z-ordering method can handle all the queries that we have listed earlier.

Range Queries: The query shape is translated into a set of z-values, as if it were a data region. Typically, we opt for an approximate representation of it, trying to balance the number of z-values and the amount of extra area in the approximation [Ore90]. Then, we search the B^+-tree with the z-values of the data regions, for matching z-values. Orenstein and Manola [OM88] describe in detail the conditions for matching.

Nearest neighbor queries: The sketch of the basic algorithm is as follows: Given a query point P, we compute its z-value and search the B^+-tree for the closest z-value; we compute the actual distance r, and then issue a range query centered at P with radius r.

Spatial joins: The algorithm for spatial joins is a generalization of the algorithm for the range query. Let S be a set of spatial objects (eg., lakes) and R be another set (eg., railways line segments). The spatial join '*find all the railways that cross lakes*' is handled as follows: the elements of set S are translated into z-values, sorted; the elements of set R are also translated into a sorted list of z-values; the two lists of z-values are merged. The details are in [Ore86] [OM88].

5.1.4 Variations - improvements

We have seen that if we traverse the pixels on ascending z-value order, we obtain a trail as shown in Figure 5.2. This trail imposes a mapping from n-d space onto a 1-d space; ideally, we would like a mapping with distance preserving properties, that is, pixels that are near in address space should have nearby z-values. The reason is that good clustering will make sure that 'similar' pixels will end up in the same or nearby leaf pages of the B^+-tree, thus greatly accelerating the retrieval on range queries.

The z-ordering indeed imposes a good such mapping: It does not leave a quadrant, unless it has visited all its pixels. However, it has some long, diagonal jumps, which maybe could be avoided. This observation prompted the search for better space filling curves. Alternatives included a curve using Gray codes [Fal88]; the best performing one is the Hilbert curve [FR89b], which has been shown to achieve better clustering than the z-ordering and the gray-codes curve, and it is the only one that we shall describe.

Figure 5.5 shows the Hilbert curves of order 1, 2 and 3: The order k curve is derived from the original, order 1 curve, by substituting each of its four points

with an order (k-1) curve, appropriately rotated or reflected. In the limit, the resulting curve has fractal dimension=2 [Man77], which intuitively means that it is so inter-twined and dense that it 'behaves' like a 2-d object. Notice also that the trail of a Hilbert curve does *not* have any abrupt jumps, like the z-ordering does. Thus, intuitively it is expected to have better distance-preserving properties than the z-ordering. Experiments in [FR89b] showed that the claim holds for the reported settings.

Algorithms to compute the Hilbert value of an n-d pixel have been published [Bia69, But71]; source code in the 'C' programming language is available in [Jag90a] for n=2 dimensions. The complexity of all these algorithms, as well as their inverses, is $O(b)$ where b is the total number of bits of the z/Hilbert value. The proportionality constant is small (a few operations per bit for the z-value, a few more for the Hilbert value). For both curves, the time to compute a z/Hilbert value is negligible compared to the disk access time.

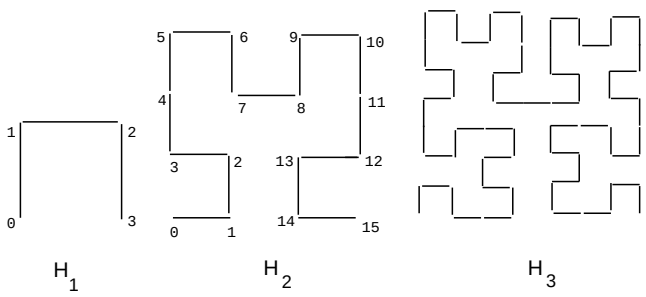


Figure 5.5 *Hilbert Curves of order 1,2 and 3*

There are several analytical and simulation studies of space filling curves: in [FR89b] we used exhaustive enumeration to study the clustering properties of several curves, showing that the Hilbert curve is best; Jagadish [Jag90a] provides analysis for partial match and 2×2 range queries; in [RF91] we derive closed formulas for the z-ordering; Moon et al. [MJFS96] derive closed formulas for range queries on the Hilbert curve.

Also related is the analytical study for quadtrees, trying to determine the number of quadtree blocks that a spatial object will be decomposed into [HS79], [Dye82, Sha88], [Fal92a], [FJM94], [Gae95], [FG96]. The common observation is that the number of quadtree blocks and the number of z/Hilbert values that a spatial object requires is *proportional to the measure of its boundary* (eg., perimeter for 2-d objects, surface area for 3-d etc.). As intuitively expected,

the constant of proportionality is smaller for the Hilbert curve, compared to the z-ordering.

5.2 R-TREES

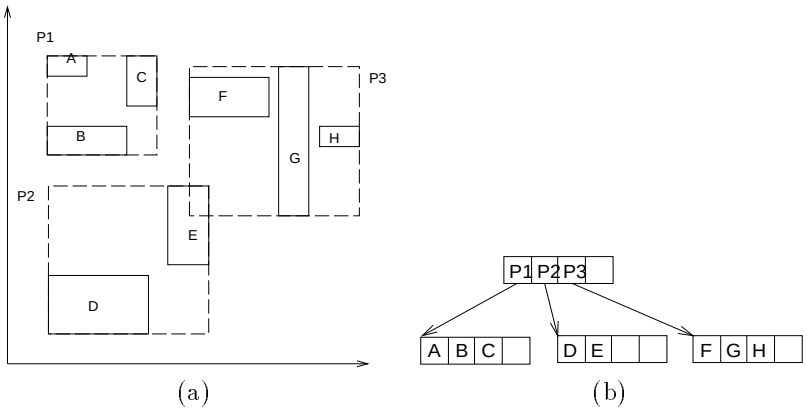


Figure 5.6 (a) Data (solid-line rectangles) organized in an R-tree with fanout=4 (b) the resulting R-tree, on disk.

The R-tree was proposed by Guttman [Gut84]. It can be thought of as an extension of the B-tree for multidimensional objects. A spatial object is represented by its minimum bounding rectangle (MBR). Non-leaf nodes contain entries of the form (ptr, R), where ptr is a pointer to a child node in the R-tree; R is the MBR that covers all rectangles in the child node. Leaf nodes contain entries of the form ($obj-id$, R) where $obj-id$ is a pointer to the object description, and R is the MBR of the object. The main innovation in the R-tree is that parent nodes are allowed to overlap. This way, the R-tree can guarantee good space utilization and remain balanced. Figure 5.6(a) illustrates data rectangles (solid boundaries), organized in an R-tree with fanout 3, while Figure 5.6(b) shows the file structure for the same R-tree, where nodes correspond to disk pages.

The R-tree inspired much subsequent work, whose main focus was to improve the search time. A packing technique is proposed in [RL85] to minimize the overlap between different nodes in the R-tree for static data. The idea was to order the data in, say, ascending x-low value, and scan the list, filling each leaf node to capacity. Figure 5.7(a) illustrates 200 points in 2-d, and Figure 5.7(b) shows the resulting R-tree parents according to the x-low sorting. An improved packing technique based on the Hilbert Curve is proposed in [KF93]: the idea

is to sort the data rectangles on the Hilbert value of their centers. Figure 5.7(c) shows the resulting R-tree parents; notice that their MBRs are closer to a square shape, which was shown to give better search performance than the elongated shapes of Figure 5.7(b).

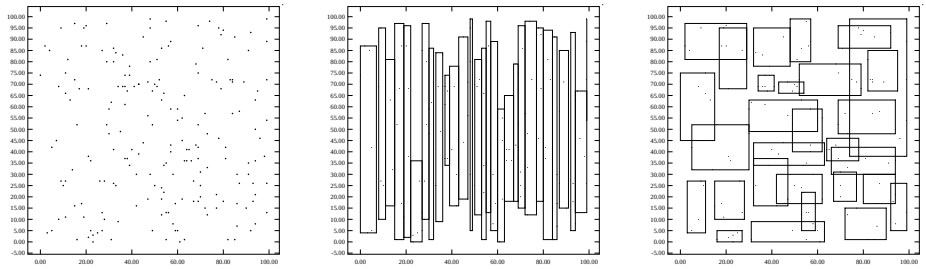


Figure 5.7 Packing algorithms for R-trees: (a) 200 points (b) their parent MBRs, when sorting on x-low (c) the parent MBRs when sorting on the Hilbert value.

A class of variations consider more general minimum bounding shapes, trying to minimize the 'dead space' that an MBR may cover. Gunther proposed the cell trees [Gun86], which introduce diagonal cuts in arbitrary orientation. Jagadish proposed the polygon trees (P-trees) [Jag90b], where the minimum bounding shapes are polygons with slopes of sides 0, 45, 90, 135 degrees. Minimum bounding shapes that are concave or even have holes have been suggested, eg., in the hB-tree [LS90].

One of the most successful ideas in R-tree research is the idea of deferred splitting: Beckmann et. al. proposed the R^*-tree [BKSS90], which was reported to outperform Guttman's R-trees by $\approx$30%. The main idea is the concept of *forced re-insert*, which tries to defer the splits, to achieve better utilization: When a node overflows, some of its children are carefully chosen and they are deleted and re-inserted, usually resulting in a better structured R-tree. The idea of deferred splitting was also exploited in the Hilbert R-tree [KF94]; there, the Hilbert curve is used to impose a linear ordering on rectangles, thus defining who the sibling of a given rectangle is, and subsequently applying the 2-to-3 (or s-to-$(s+1)$) splitting policy of the B^*-trees (see section 3.2). Both methods achieve higher space utilization than Guttman's R-trees, as well as better response time (since the tree is shorter and more compact).

Finally, analysis of the R-tree performance has attracted a lot of interest: in [FSR87] we provide formulas, assuming that the spatial objects are uniformly distributed in the address space. The uniformity assumption was relaxed in [FK94], where we showed that the *fractal dimension* is an excellent measure of the non-uniformity, and we provided accurate formulas to estimate the average number of disk accesses of the resulting R-tree. The fractal dimension also helps estimate the selectivity of spatial joins, as shown in [BF95]. When the sizes of the MBRs of the R-tree are known, formulas for the expected number of disk access are given in [PSTW93] and [KF93].

5.2.1 Algorithms

Since the R-tree is one of the most successful spatial access methods, we describe the related algorithms in some more detail:

Insertion: When a new rectangle is inserted, we traverse the tree to find the most suitable leaf node; we extend its MBR if necessary, and store the new rectangle there. If the leaf node overflows, we split it, as discussed next:

Split: This is one of the most crucial operations for the performance of the R-tree. Guttman suggested several heuristics to divide the contents of an overflowing node into two sets, and store each set in a different node. Deferred splitting, as mentioned in the R^*-tree and in the Hilbert R-tree, will improve performance. Of course, as in B-trees, a split may propagate upwards.

Range Queries: The tree is traversed, comparing the query MBR with the MBRs in the current node; thus, non-promising (and potentially large) branches of the tree can be 'pruned' early.

Nearest Neighbors: The algorithm follows a 'branch-and-bound' technique similar to [FN75] for nearest-neighbor searching in clustered files. Roussopoulos et al. [RKV95] give the detailed algorithm for R-trees.

Spatial Joins: Given two R-trees, the obvious algorithm builds a list of pairs of MBRs, that intersect; then, it examines each pair in more detail, until we hit the leaf level. Significantly faster methods than the above straightforward method have been suggested in [BKS93] [BKSS94]. Lo and Ravishankar [LR94] proposed an efficient method to perform a spatial join when only one of the two spatial datasets has an R-tree index on it.

5.2.2 Conclusions

R-trees is one of the most promising SAMs. Among its variations, the R^*-trees and the Hilbert R-trees seem to achieve the best response time and space utilization, in exchange for more elaborate splitting algorithms.

5.3 TRANSFORMATION TO HIGHER-D POINTS

The idea is to transform 2-d rectangles into 4-d points [HN83], by using the low- and high-values for each axis; then, any point access method (PAM) can be used. In general, an n-dimensional rectangle will become a $2n$-dimensional point. The original and final space are called 'native' and 'parameter' space, respectively [Ore90]. Figure 5.8 illustrates the idea for 1-d address space: line segments A(0:0.25) and B(0.75:1) are mapped into 2-d points. A range query Q(0.25:0.75) in native space becomes a range query in parameter space, as illustrated by the shaded region in Figure 5.8.

The strong point of this idea is that we can turn any Point Access Method (PAM) into a Spatial Access Method (SAM) with very little effort. This approach has been used or suggested in several settings, eg., with grid files [HN83], B-trees [FR91], hB-trees [LS90] as the underlying PAM.

The weak points are the following: (a) the parameter space has high dimensionality, inviting 'dimensionality curse' problems earlier on (see the discussion on page 39). (b) except for range queries, there are no published algorithms for nearest-neighbor and spatial join queries. Nevertheless, it is a clever idea, which can be valuable for a stand-alone, special purpose system, operating on a low-dimensionality address space. Such an application could be, eg., a temporal database system [SS88], where the spatial objects are 1-dimensional time segments [KTF95].

5.4 CONCLUSIONS

From a practical point of view, the most promising methods seem to be:

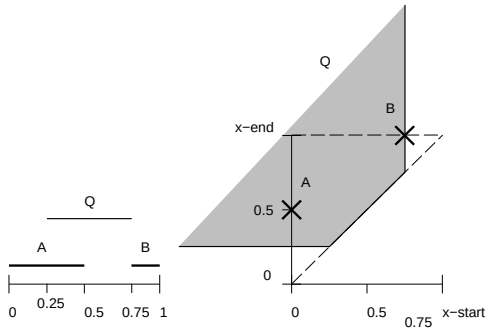


Figure 5.8 Transformation of 1-d rectangles into points in higher dimensionality.

- Z-ordering: Z-ordering and, equivalently, linear quadtrees have been very popular for 2-dimensional spaces. One of the major application is in geographic information systems: linear quadtrees have been used both in production systems, like the TIGER system at the U.S. Bureau of Census [Whi81] (`http://tiger.census.gov/tiger/tiger.html`), which stores the map and statistical data of the U.S.A., as well as research prototypes such as QUILT [SSN87], PROBE [OM88], and GODOT [GR94]. For higher dimensions, oct-trees have been used in 3-d graphics and robotics [BB82]; in databases of 3-d medical images [ACF+94], etc. Z-ordering performs very well for a low dimensionality and for points. It is particularly attractive because it can be implemented on top of a B-tree with relatively little effort. However, for objects with non-zero area (= hyper-volume), the practitioner should be aware of the fact that each such object may require a large number of z-values for its exact representation; the recommended approach is to approximate each object with a small number of z-values [Ore89, Ore90].

- R-trees and variants: they operate on the native space (requiring no transforms to high-dimensionality spaces), and they can handle rectangles and other shapes without the need to divide them into pieces. Cutting data into pieces results in an artificially increased database size (linear on the number of *pieces*); moreover, it requires a duplicate-elimination step, because a query may retrieve the same object-id several times (once for each piece of the qualifying object) R-trees have been tried successfully for 20-30 dimensional address spaces [FBF+94, PF94]. Thanks to the above properties, R-trees have been incorporated in academic as well as commercial sys-

tems, like POSTGRES (`http://s2k-ftp.cs.berkeley.edu:8000/postgres95/`) and ILLUSTRA (`http://www.illustra.com/`).

Before we close this chapter, we should mention about the 'dimensionality curse'. Unfortunately, all the SAMs will suffer for high dimensionalities n: For the z-ordering, the range queries of radius r will require effort proportional to the hyper-surface of the query region $O(r^{(n-1)})$ as mentioned on page 33. Similarly, for the R-trees as the dimensionality n grows, each MBR will require more space; thus, the fanout of each R-tree page will decrease. This will make the R-tree taller and slower. However, as mentioned, R-trees have been successfully used for 20-30 dimensions [FBF+94] [PF94]. To the best of this author's knowledge, performance results for the z-ordering method are available for low dimensionalities only (typically, n=2). A comparison of R-trees versus z-ordering for high dimensionalities is an interesting research question.

Exercises

Exercise 5.1 [07] *What is the z-value of the pixel (`11, 00`) in Figure 5.2? What is its Hilbert value?*

Exercises

Exercise 5.2 [20] *Design an algorithm for the spatial join in R-trees*

Exercise 5.3 [30] *Design an algorithm for the k nearest neighbors in R-trees*

Exercise 5.4 [30] *Repeat the two previous exercises for the Z-ordering*

Exercise 5.5 [38] *Implement the code for the Hilbert curve, for 2 dimensions; for n dimensions.*

6

ACCESS METHODS FOR TEXT

6.1 INTRODUCTION

In this chapter we present the main ideas for text retrieval methods. For more details, see the survey in [Fal85] and the book by Frakes and Baeza-Yates [FBY92]. Access methods for text are interesting for three reasons: (a) multimedia objects often have captions in free text; exploiting these captions may help retrieve some additional relevant objects [OS95] (b) research in text retrieval has led to some extremely useful ideas, like the *relevance feedback* and the *vector space* model that we discuss next and (c) text retrieval has several applications in itself.

Such applications include the following:

- Library automation [SM83] [Pri84] and distributed digital libraries [GGMT94], where large amounts of text data are stored on the world-wide-web (WWW). In this setting, search engines are extremely useful and popular, like 'veronica' [ODL93], 'lycos' (`http://lycos.cs.cmu.edu/`), 'inktomi' (`http://inktomi.berkeley.edu/`), etc..
- Automated law and patent offices [Hol79] [HH83]; electronic office filing [CTH+86]; electronic encyclopedias [EMS+86] and dictionaries [GT87].
- Information filtering (eg., the *RightPages* project [SOF+92] and the *Latent Semantic Indexing* project [DDF+90] [FD92a]); also, the 'selective dissemination of information' (SDI) [YGM94].

In text retrieval, the queries can be classified as follows [Fal85]:

- Boolean queries, eg. '*(data* `or` *information)* `and` *retrieval* `and` *(*`not` *text)*'. Here, the user specifies terms, connected with Boolean operators. Some additional operators are also supported, like `adjacent`, or `within` <*n*> `words` or `within sentence`, with the obvious meanings. For example the query '*data* `within sentence` *retrieval*' will retrieve documents which contain a sentence with both the words 'data' and 'retrieval'.
- Keyword search: here, the user specifies a set of keywords, like, eg., '*data, retrieval, information*'; the retrieval system should return the documents that contain as many of the above keywords as possible. This interface offers less control to the user, but it is more user-friendly, because it does not require familiarity with Boolean logic.

Several systems typically allow for prefix matches, eg., '*organ**' will match all the words that start with 'organ', like 'organs', 'organization', 'organism' etc. We shall use the star '*' as the variable-length don't care character.

The rest of this chapter is organized as follows: in the next three sections we discuss the main three methods for text retrieval, namely (a) full text scanning, (b) inversion and (c) signature files. In the last section we discuss the clustering approach.

6.2 FULL TEXT SCANNING

According to this method, no preprocessing of the document collection is required. When a query arrives, the whole collection is inspected, until the matching documents are found.

When the user specifies a pattern that is a *regular expression*, the textbook approach is to use a finite state automaton (FSA) [HU79, pp. 29-35]. If the search pattern is a single string with no *don't care* characters, faster methods exist, like the Knuth, Morris and Pratt algorithm [KMP77], and the fastest of all, the Boyer and Moore algorithm [BM77] and its recent variations [Sun90, HS91].

For multiple query strings, the algorithm by Aho and Corasick [AC75] builds a finite state automaton in time linear on the total length of the strings, and reports all the matches in a single pass over the document collection.

Searching algorithms that can tolerate typing errors have been developed by Wu and Manber [WM92] and Baeza-Yates and Gonnet [BYG92]. The idea is to scan the database one character at a time, keeping track of the currently matched characters. The algorithm can retrieve all the strings within a desired *editing distance* from the query string. The editing distance of two strings is the minimum number of insertions, deletions and substitutions that are needed to transform the first string into the second [HD80, LW75, SK83]. The method is flexible and fast, requiring a few seconds for a few Megabytes of text on a SUN-class workstation. Moreover, its source code is available (`ftp://cs.arizona.edu/agrep`).

In general, the advantage of every full text scanning method is that it requires no space overhead and minimal effort on insertions and updates, since no indices have to be changed. The price is that the response time is slow for large data bases. Therefore, full text scanning is typically used for small databases (a few Mbytes in size), or in conjunction with another access method (e.g., inversion) that would restrict the scope of searching.

6.3 INVERSION

In inversion, each document can be represented by a list of (key)words, which describe the contents of the document for retrieval purposes. Fast retrieval can be achieved if we invert on those keywords: The keywords are stored, eg., alphabetically, in the 'index file'; for each keyword we maintain a list of pointers to the qualifying documents in the 'postings file'. Figure 6.1 illustrates the file structure, which is very similar to the inverted index for secondary keys (see Figure 4.1).

Typically, the index file is organized using sophisticated primary-key access methods, such as B-trees, hashing, TRIEs [Fre60] or variations and combinations of these (e.g., see [Knu73, pp. 471-542], or Chapter 3). For example, in an early version of the UNIX^{TM} utility `refer`, Lesk used an over-loaded hash table with separate chaining, in order to achieve fast searching in a database of bibliographic entries [Les78]; the Oxford English Dictionary uses an extension of the PATRICIA trees [Mor68], called PAT trees [GBYS92].

The advantages are that inversion is relatively easy to implement, it is fast, and it supports synonyms easily (e.g., the synonyms can be organized as a threaded list within the dictionary). For the above reasons, the inversion method has

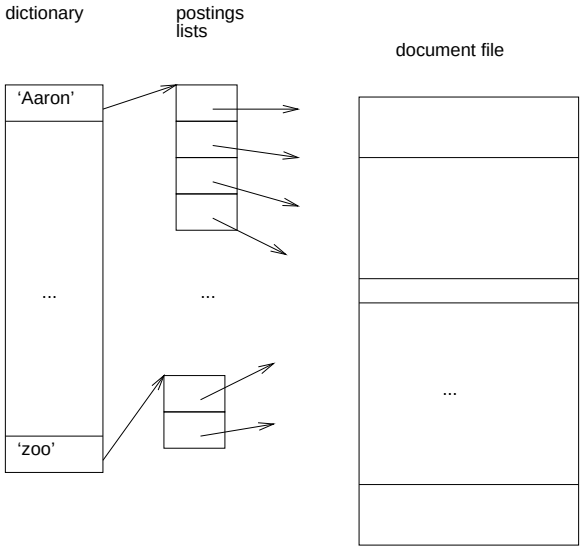


Figure 6.1 Illustration of inversion

been adopted in most of the commercial systems such as DIALOG, BRS, MEDLARS, ORBIT, STAIRS [SM83, ch. 2].

The disadvantages of this method are the storage overhead (which can reach up to 300% of the original file size [Has81] if too much information is kept about each word occurrence), and the cost of updating and reorganizing the index, if the environment is dynamic.

Recent work exactly focuses on these problems: Techniques to achieve fast insertions incrementally include the work by Tomasic et al., [TGMS94]; Cutting and Pedersen [CP90] and Brown et. al. [BCC94]. These efforts typically exploit the skewness of the distribution of postings lists, treating the short lists differently than the long ones.

It is important to elaborate on the skewness, because it has serious implications for the design of fast text retrieval methods. The distribution typically follows *Zipf's law* [Zip49], which states that the occurrence frequency of a word is inversely proportional to its rank (after we sort the vocabulary words in

decreasing frequency order). More specifically, we have [Sch91]:

$$f(r) = \frac{1}{r \ln(1.78V)} \tag{6.1}$$

where r is the rank of the vocabulary word, $f(r)$ is the percentage of times it appears, and V is the vocabulary size. This means that a few vocabulary words will appear very often, while the majority of vocabulary words will appear once or twice. Figure 6.2 plots the frequency versus the rank of the words in the Bible, in logarithmic scales. The Figure also plots the predictions, according to Eq. 6.1. Notice that the first few most common words appear tens of thousands of times, while the vast majority of the vocabulary words appear less than 10 times. Zipf reported that similar skeweness is observed in several other languages, in addition to English.

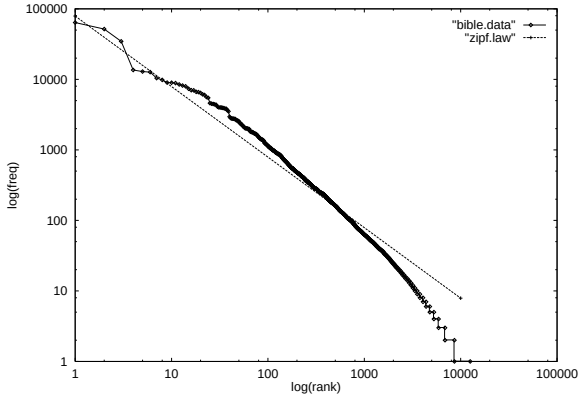


Figure 6.2 Rank-Frequency plot for the words in the Bible - both scales are logarithmic. The line corresponds to Zipf's law.

Given the large size of indices, compression methods have also been investigated: Zobel et al. [ZMSD92] use Elias's [Eli75] compression scheme for postings lists, reporting small space overheads for the index. Finally, the `glimpse` package [MW94] uses a coarse index plus the `agrep` package [WM92] for approximate matching. Like the `agrep` package, `glimpse` is also available from the University of Arizona (`ftp: //cs.arizona.edu/ glimpse`).

6.4 SIGNATURE FILES

The idea behind signature files is to create a 'quick and dirty' filter, which will be able to quickly eliminate most of the non-qualifying documents. As we shall see next and in Chapter 7, this idea has been used several times in very different contexts, often with excellent results. A recent survey on signature files is in [Fal92b].

The method works as illustrated in Figure 6.3: For every document, a short, typically hash-coded version of it is created (its *document signature*); document signatures are typically stored sequentially, and are searched upon a query. The signature test returns *all* the qualifying documents, plus (hopefully, few) false matches, or '*false alarms*' or '*false drops*'. The documents whose signatures qualify are further examined, to eliminate the false drops.

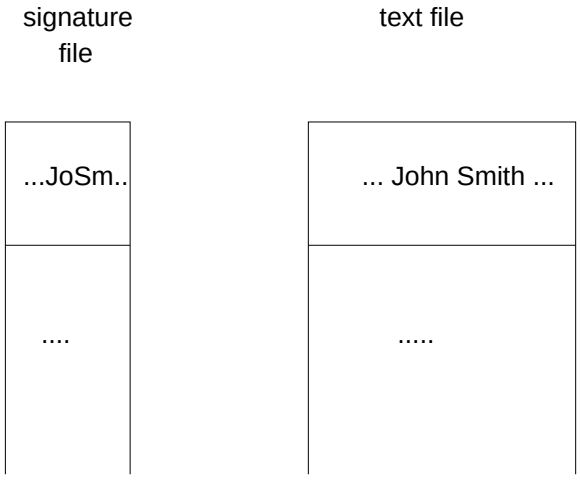


Figure 6.3 Example of signature files. For illustration, the signature of a word is decided to be its first two letters.

Figure 6.3 shows a naive (and not recommended) method of creating signatures, namely, by keeping the first 2 letters of every word in the document. One of the best methods to create signatures is *superimposed coding* [Moo49]. Following the notation in [CF84], each word yields a bit pattern (*word signature*) of size F, with m bits set to '`1`' and the rest left as '`0`'. These bit patterns are OR-ed to form the document signature. Table 6.3 gives an example for a toy document, with two words: '*data*' and '*base*'.

Word	Signature
data	001 000 110 010
base	000 010 101 001
doc. signature	001 010 111 011

Table 6.1 Illustration of superimposed coding: F=12 bits for a signature, with m=4 bits per word set to 1.

On searching for a word, say '*data*', we create its word signature, and exclude all the document signatures that do not have a '1' at the corresponding bit positions. The choice of the signature length F depends on the desirable false-drop rate; the m parameter is chosen such that, on the average, half of the bits should be '1' in a document signature [Sti60].

The method has been studied and used extensively in text retrieval: on bibliographic entries [FH69]; for fast substring matching [Har71] [KR87]; in office automation and message filing [TC83] [FC87]. It has been used in academic [SDR83, SDKR87] as well as commercial systems [SK86, Kim88].

Moreover, signature files with some variation of superimposed coding have been used in numerous other applications: For indexing of formatted records [Rob79, CS89]; for indexing of images [RS92]; for set-membership testing in the so-called *Bloom filters*, which have been used for spelling checking in UNIX™ [McI82], in differential files [SL76], and in semi-joins for distributed query optimization [ML86]. A variation of superimposed coding has even been used in chess-playing programs, to alert for potentially dangerous combinations of conditions [ZC77].

In concluding this discussion on the signature file approach, its advantages are the simplicity of its implementation, and the efficiency in handling insertions. In addition, the method is trivially parallelizable [SK86]. The disadvantage is that it may be slow for large databases, unless the sequential scanning of the signature file is somehow accelerated [SDR83, LL89, ZRT91].

6.5 VECTOR SPACE MODEL AND CLUSTERING

The Vector Space Model is very popular in information retrieval [Sal71b] [SM83] [VR79], and it is well suited for 'keyword queries'. The motivation behind the approach is the so-called cluster hypothesis: closely associated documents tend to be relevant to the same requests. Grouping similar documents accelerates the searching.

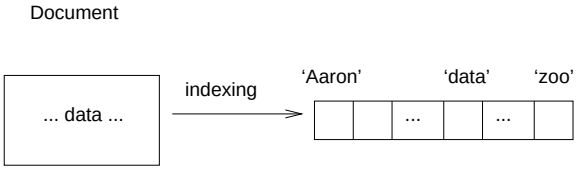


Figure 6.4 Illustration of the 'indexing' process in IR

An important contribution of the vector space model is to envision each document as a V-dimensional vector, where V is the number of terms in the document collection. The procedure of mapping a document into a vector is called *indexing* (overloading the word!); 'indexing' can be done either manually (by trained experts), or automatically, using a stop list of common words, some stemming algorithm, and possibly a thesaurus of terms. The final result of the 'indexing' process is that each document is represented by a V-dimensional vector, where V is the number of permissible index terms. Absence of a term is indicated by a 0 (or by -1 [Coo70]); presence of a term is indicated by 1 (for binary document vectors) or by a positive number (term weight), which reflects the importance of the term for the document.

The next step in the vector space model is to decide how to group similar vectors together ('*cluster generation*'); the last step is to decide how to search a cluster hierarchy for a given query ('*cluster search*'). For both the above problems, we have to decide on a *document-to-document similarity function* and on a *document-to-cluster similarity function*. For the document-to-document similarity function, several choices are available, with very similar performance ([SM83, pp. 202-203] [VR79, p. 38]). Thus, we present only the *cosine similarity function* [Sal71b] which seems to be the most popular:

$$\cos(\vec{x}, \vec{y}) = \vec{x} \circ \vec{y} / (\| \vec{x} \| \, \| \vec{y} \|) \tag{6.2}$$

where $\vec{x}$ and $\vec{y}$ are two V-dimensional document vectors, '$\circ$' stands for the inner product of two vectors and $\| \, . \, \|$ for the Euclidean norm of its argument.

There are also several choices for the document-to-cluster distance/similarity function. The most popular seems to be the method that treats the centroid of the cluster as a single document, and then applies the document-to-document similarity/distance function. Other choices include the 'single link' method, which estimates the minimum distance (= dis-similarity) of the document from all the members of the cluster, and the 'all link' method which computes the maximum of the above distances.

An interesting and effective recent development is the 'Latent Semantic Indexing' (LSI), which applies the Singular Value Decomposition (SVD) on the document-term matrix and it automatically groups co-occurring terms. These groups can be used as a thesaurus, to expand future queries. Experiments [FD92b] showed up to 30% improvement over the traditional vector model. More details on the method are in Appendix D.3, along with the description of the SVD.

In the next subsections we briefly describe the main ideas behind (a) the cluster generation algorithms, (b) the cluster search algorithms and (c) the evaluation methods of the clustering schemes.

6.5.1 Cluster generation

Several cluster generation methods have been proposed; recent surveys can be found in [Ras92] [Mur83] [VR79]. Following Van-Rijsbergen [VR79], we distinguish two classes:

- 'sound' methods, that are based on the document-to-document similarity matrix and
- 'iterative' methods, that are more efficient and proceed directly from the document vectors.

Sound Methods: If N is the number of documents, these methods usually require $O(N^2)$ time (or more) and apply graph theoretic techniques. A simplified version of such a clustering method would work as follows ([DH73b, p. 238]): An appropriate threshold is chosen and two documents with a similarity measure that exceeds the threshold are assumed to be connected with an edge. The connected components (or the maximal cliques) of the resulting graph are the proposed clusters.

The problem is the selection of the appropriate threshold: different values for the threshold give different results. The method proposed by Zahn [Zah71] is an attempt to circumvent this problem. He suggests finding a minimum spanning tree for the given set of points (documents) and then deleting the 'inconsistent' edges. An edge is inconsistent if its length l is much larger than the average length l_{avg} of its incident edges. The connected components of the resulting graph are the suggested clusters. Figure 6.5 gives an illustration of the method. Notice that the long edges with solid lines are *not* inconsistent, because, although long, they are not significantly longer than their adjacent edges.

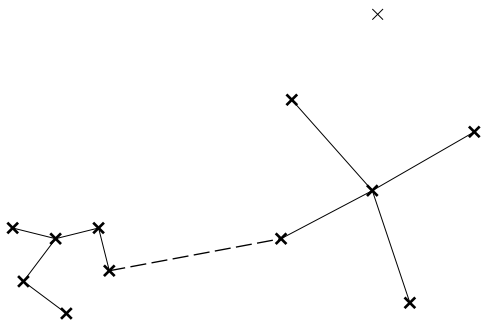


Figure 6.5 Illustration of Zahn's method: the dashed edge of the MST is 'inconsistent' and therefore deleted; the connected components are the clusters.

Iterative methods: This class consists of methods that are faster: $O(N \log N)$ or $O(N^2/\log N)$) on the average. They are based directly on the object (document) descriptions and they do not require the similarity matrix to be computed in advance. The typical iterative method works as follows:

- Choose some seeds (eg., from sound clustering on a sample)
- Assign each vector to the closest seed (possibly adjusting the cluster centroid)
- Possibly, re-assign some vectors, to improve clusters

Several iterative methods have been proposed along these lines, The simplest and fastest one seems to be the 'single pass' method [SW78]: Each document is processed once and is either assigned to one (or more, if overlap is allowed) of the existing clusters, or it creates a new cluster.

In conclusion, as mentioned before, the iterative ones are fast and practical, but they are sensitive to the insertion order.

6.5.2 Cluster searching

Searching in a clustered file is much simpler than cluster generation. The input query is represented as a V-dimensional vector and it is compared with the cluster-centroids. The searching proceeds in the most similar clusters, i.e., those whose similarity with the query vector exceeds a threshold. As mentioned, a typical cluster-to-query similarity function is the cosine function - see Eq. 6.2

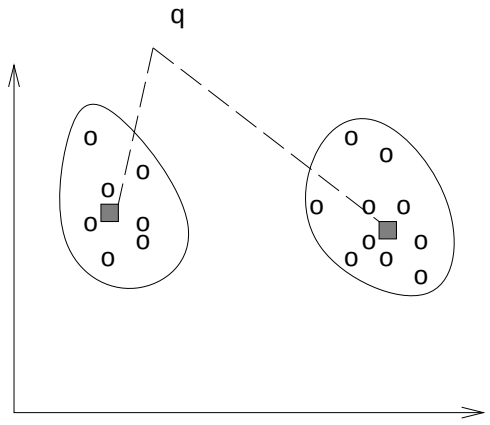


Figure 6.6 Searching in clustered files: For the query vector q, searching continues in the closest cluster at the left.

The vector representation of queries and documents has led to two important ideas:

- *ranked output* and
- *relevance feedback*

The first idea, ranked output, comes naturally, because we can always compute the distance/similarity of the retrieved documents to the query, and we can sort them ('most similar first') and present only the first screenful to the user.

The second idea, the relevance feedback, is even more important, because it can easily increase the effectiveness of the search [Roc71]: The user pinpoints the relevant documents among the retrieved ones and the system re-formulates the query vector and starts the searching from the beginning. To carry out the query re-formulation, we operate on the query vector and add (vector addition) the vectors of the relevant documents and subtract the vectors of the non-relevant ones. Experiments indicate that the above method gives excellent results after only two or three iterations [Sal71a].

6.5.3 Evaluation of clustering methods

The standard way to evaluate the 'goodness' of a clustering method is to use the so-called *precision* and *recall* concepts. Thus, given a collection of documents, a set of queries and a human expert's responses to the above queries, the ideal system is the one that will retrieve exactly what the human dictated, and nothing more. The deviations from the above ideal situation are measured by the precision and recall: consider the set of documents that the computerized system returned; then the precision is defined as the percentage of relevant documents among the retrieved ones:

$$\text{precision} \equiv \frac{\text{retrieved and relevant}}{\text{retrieved}}$$

and recall is the percentage of relevant documents that we retrieved, over the total number of relevant documents in the document collection:

$$\text{recall} \equiv \frac{\text{retrieved and relevant}}{\text{relevant}}$$

Thus, high precision means that we have few false alarms; high recall means that we have few false dismissals.

The popular precision-recall plot gives the scatter-plot of the precision-recall values for several queries. Figure 6.7 shows such a plot, for fictitious data. When comparing the precision-recall plots of competing methods, the one closer to the (1.0,1.0) point is the winner.

The annual *Text REtrieval Conference* (TREC) provides a test-bed for an open competition of text retrieval methods. See `http: //potomac. ncsl. nist.gov/ TREC/` for more details.

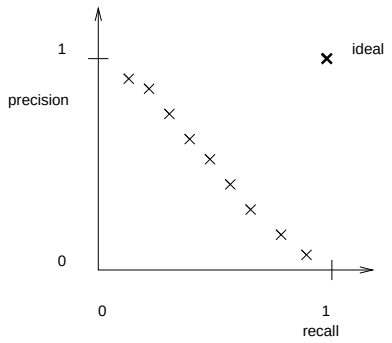


Figure 6.7 A fictitious, typical recall-precision diagram

6.6 CONCLUSIONS

We have discussed several methods for text searching and information retrieval (IR). Among them, the conclusions for a practitioner are as follows:

- Full text scanning is suitable for small databases (up to a few Megabytes); '`agrep`' is a recent, excellent free-ware search package. Its scaled-up version, '`glimpse`', allows fast searching for up to a few Gigabytes of size.
- Inversion is the industry work-horse, for larger databases.
- Signatures can provide a 'quick-and-dirty' test, when false alarms are tolerable or when they can be quickly discarded.
- The major ideas from the vector space model are two: (a) the relevance feedback and (b) the ability to provide ranked output (i.e., documents sorted on relevance order)

Exercises

Exercise 6.1 [20] *Produce the (rank, frequency) plot for a text file. (Hint: use the commands* `sort -u`, `tr` *and* `awk` *from* UNIX™ *).*

Exercise 6.2 [30] *For queries with a single string and no 'don't care' characters, implement the straightforward method for full text scanning; also, type-in*

the Boyer-Moore code from Sunday [Sun90]; time them on some large files; compare them to `grep` *and* `agrep`*.*

Exercise 6.3 [25] *Implement an algorithm that will compute the editing distance of two strings, that is, the minimum number of insertions, deletions or substitutions that are needed to transform one string to the other. (Hint: see the discussion by Hall and Dowling [HD80]).*

Exercise 6.4 [40] *Develop a B-tree package and use it to build an index for text files.*

Exercise 6.5 [40] *Develop and implement algorithms to do insertion and search in a cluster hierarchy, as described in [SW78].*

PART II

INDEXING SIGNALS

7

PROBLEM - INTUITION

7.1 INTRODUCTION

The problem we focus on is the design of fast searching methods that will search a database of multimedia objects, to locate objects that match a query object, exactly or approximately. Objects can be 2-dimensional color images, gray-scale medical images in 2-d or 3-d (eg., MRI brain scans), 1-dimensional time sequences, digitized voice or music, video clips etc. A typical query by content would be, eg., '*in a collection of color photographs, find ones with a same color distribution as a sunset photograph*'.

Specific applications include image databases; financial, marketing and production time sequences; scientific databases with vector fields; audio and video databases, DNA/genome databases, etc. In such databases, typical queries would be '*find companies whose stock prices move similarly*', or '*find images that look like a sunset*', or '*find medical X-rays that contain something that has the texture of a tumor*'.

Searching for similar patterns in such databases as the above is essential, because it helps in predictions, decision making, computer-aided medical diagnosis and teaching, hypothesis testing and, in general, in 'data mining' [AGI+92, AIS93b] [AIS93a, AS94, HS95] and rule discovery.

The first important step is to provide a measure for the distance between two objects. We rely on a domain expert to supply such a distance function $\mathcal{D}()$:

Definition 7.1 *Given two objects, O_A and O_B, the distance (= dis-similarity) of the two objects is denoted by*

$$\mathcal{D}(O_A, O_B) \tag{7.1}$$

For example, if the objects are two (equal-length) time sequences, the distance $\mathcal{D}()$ could be their Euclidean distance (sum of squared differences, see Eq. 7.2).

Similarity queries can been classified into two categories:

Whole Match: Given a collection of N objects $O_A, O_B, \ldots, O_N$ and a query object Q, we want to find those data objects that are within distance ϵ from Q. Notice that the query and the objects are of the same type: for example, if the objects are 512×512 gray-scale images, so is the query.

Sub-pattern Match: Here the query is allowed to specify only part of the object. Specifically, given N data objects (eg., images) $O_A, O_B, \ldots, O_N$, a query (sub-)object Q and a tolerance ϵ, we want to identify the parts of the data objects that match the query. If the objects are, eg., 512×512 gray-scale images (like medical X-rays), in this case the query could be, eg., a 16×16 sub-pattern (eg., a typical X-ray of a tumor).

Additional types of queries include the '*nearest neighbors*' queries (eg., 'find the 5 most similar stocks to IBM's stock') and the '*all pairs*' queries or '*spatial joins*' (eg., '*report all the pairs of stocks that are within distance ϵ from each other*'). Both the above types of queries can be supported by our approach: As we shall see, we reduce the problem into searching for multi-dimensional points, which will be organized in R-trees; in this case, we know of algorithms for both nearest-neighbor search as well as spatial joins, as discussed in Chapter 5. Thus, we do not focus on nearest-neighbor and 'all-pairs' queries.

For all the above types of queries, the ideal method should fulfill the following requirements:

- It should be *fast*. Sequential scanning and distance calculation with each and every object will be too slow for large databases.
- It should be '*correct*'. In other words, it should return all the qualifying objects, without missing any (i.e., no 'false dismissals'). Notice that 'false alarms' are acceptable, since they can be discarded easily through a post-processing step. Of course, as we see, eg., in Figure 8.2, we try to keep their number low, so that the total response time is minimized.

- The proposed method should require a small space overhead.
- The method should be dynamic. It should be easy to insert, delete and update objects.

As we see next, the heart of the proposed approach is to use k feature extraction functions, to map objects into points in k-dimensional space; thus, we can use highly fine-tuned database spatial access methods to accelerate the search. In the next Chapter we describe the details of the main idea. In Chapters 8 and 9 we describe how this idea has been applied for time sequences and color images. Chapter 10 discusses how to extend the ideas to handle sub-pattern matching in time sequences. Chapter 11 discusses a fast, approximate method of extracting features from objects, so that the distance is preserved. Chapter 12 lists the conclusions for this Part.

7.2 BASIC IDEA

To illustrate the basic idea, we shall focus on 'whole match' queries. There, the problem is defined as follows:

- we have a collection of N objects: O_A, O_B, ..., O_N
- the distance/dis-similarity between two objects (O_i, O_j) is given by the function $\mathcal{D}(O_i, O_j)$, which can be implemented as a (possibly, slow) program
- the user specifies a query object Q, and a tolerance ϵ

Our goal is to find the objects in the collection that are within distance ϵ from the query object. An obvious solution is to apply sequential scanning: For each and every object O_i $(1 \leq i \leq N)$, we can compute its distance from Q and report the objects with distance $\mathcal{D}(Q, O_i) \leq \epsilon$.

However, sequential scanning may be slow, for two reasons:

1. the distance computation might be expensive. For example, the editing distance [HD80] in DNA strings requires a dynamic-programming algorithm, which grows like the product of the string lengths (typically, in the hundreds or thousands, for DNA databases).

2. the database size N might be huge.

Thus, we are looking for a faster alternative. The proposed approach is based on two ideas, each of which tries to avoid each of the two disadvantages of sequential scanning:

- a 'quick-and-dirty' test, to discard quickly the vast majority of non-qualifying objects (possibly, allowing some false-alarms)
- the use of Spatial Access Methods (SAMs), to achieve faster-than-sequential searching, as suggested by Jagadish [Jag91].

The case is best illustrated with an example. Consider a database of time sequences, such as yearly stock price movements, with one price per day. Assume that the distance function between two such sequences S and Q is the Euclidean distance

$$\mathcal{D}(S,Q) \equiv \left(\sum_{i=1}(S[i]-Q[i])^2\right)^{1/2} \tag{7.2}$$

where $S[i]$ stands for the value of stock S on the i-th day. Clearly, computing the distance of two stocks will take 365 subtractions and 365 squarings in our example.

The idea behind the 'quick-and-dirty' test is to characterize a sequence with a single number, which will help us discard many non-qualifying sequences. Such a number could be, eg., the average stock price over the year: Clearly, if two stocks differ in their averages by a large margin, it is impossible that they will be similar. The converse is not true, which is exactly the reason we may have false alarms. Numbers that contain some information about a sequence (or a multimedia object, in general), will be referred to as '*features*' for the rest of this paper. Using a good feature (like the 'average', in the stock-prices example), we can have a quick test, which will discard many stocks with a single numerical comparison for each sequence, a big gain over the 365 subtractions and squarings that the original distance function requires.

If using one feature is good, using two or more features might be even better, because they may reduce the number of false alarms (at the cost of making the 'quick-and-dirty' test a bit more elaborate and expensive). In our stock-prices example, additional features might be, eg., the standard deviation, or, even better, some of the discrete Fourier transform (DFT) coefficients, as we shall see in Chapter 8.

The end result of using k features for each of our objects is that we can map each object into a point in k-dimensional space. We shall refer to this mapping as $\mathcal{F}()$ (for 'F'eature):

Definition 7.2 *Let $\mathcal{F}()$ be the mapping of objects to k-d points, that is $\mathcal{F}(O)$ will be the k-d point that corresponds to object O.*

This mapping provides the key to improve on the second drawback of sequential scanning: by organizing these k-d points into a spatial access method, we can cluster them in a hierarchical structure, like the R-trees. Upon a query, we can exploit the R-tree, to prune out large portions of the database that are not promising. Thus, we do not even *have* to do the quick-and-dirty test on all of the k-d points!

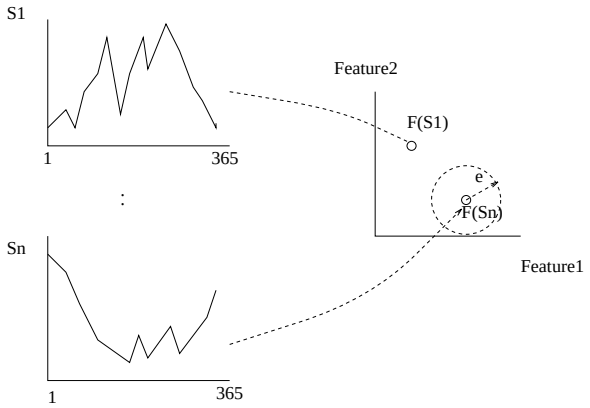


Figure 7.1 Illustration of basic idea: a database of sequences S1, ... Sn; each sequence is mapped to a point in feature space; a query with tolerance ϵ becomes a sphere of radius ϵ.

Figure 7.1 illustrates the basic idea: Objects (eg., time sequences that are 365-points long) are mapped into 2-d points (eg., using the average and the standard-deviation as features). Consider the 'whole match' query that requires all the objects that are similar to S_n within tolerance ϵ: this query becomes an k-d sphere in feature space, centered on the image $\mathcal{F}(S_n)$ of S_n. Such queries on multidimensional points is exactly what R-trees and other SAMs are designed to answer efficiently. More specifically, the search algorithm for a whole match query is illustrated in Figure 7.2.

Algorithm 7.1 Search for whole-match queries:

1. map the query object Q into a point $\mathcal{F}(Q)$ in feature space
2. using the SAM, retrieve all points within the desired tolerance ϵ from $\mathcal{F}(Q)$.
3. retrieve the corresponding objects, compute their actual distance from Q and discard the false alarms.

Figure 7.2 Pseudo-code for the search algorithm.

Intuitively, this approach has the potential to relieve both problems of the sequential scan, presumably resulting into much faster searches. The only step that we have to be careful with is that the mapping $\mathcal{F}()$ from objects to k-d points does not distort the distances. Let $\mathcal{D}()$ be the distance function of two objects, and $\mathcal{D}_{feature}()$ be the (say, Euclidean) distance of the corresponding feature vectors. Ideally, the mapping should preserve the distances exactly, in which case the SAM will have neither false alarms nor false dismissals. However, requiring perfect distance preservation might be difficult: For example, it is not obvious which features we have to use to match the editing distance between two DNA strings. Even if the features are obvious, there might be practical problems: for example, in the stock-price example, we could treat every sequence as a 365-dimensional vector; although in theory a SAM can support an arbitrary number of dimensions, in practice all SAMs suffer from the 'dimensionality curse', as discussed in Chapter 5.

The crucial observation is that we can guarantee that the proposed method will not result in any false dismissals, if the distance in feature space matches or underestimates the distance between two objects. Intuitively, this means that our mapping $\mathcal{F}()$ from objects to points *should make things look closer*, ie., it should be a contractive mapping.

Mathematically, let O_A and O_B be two objects (e.g., same-length sequences) with distance function $\mathcal{D}()$ (e.g., the Euclidean distance) and $\mathcal{F}(O_1)$, $\mathcal{F}(O_2)$ be their feature vectors (e.g., their first few Fourier coefficients), with distance function $\mathcal{D}_{feature}()$ (e.g., the Euclidean distance, again). Then we have:

Lemma 1 (Lower-bounding) *To guarantee no false dismissals for whole-match queries, the feature extraction function $\mathcal{F}()$ should satisfy the following formula:*

$$\mathcal{D}_{feature}(\mathcal{F}(O_1), \mathcal{F}(O_2)) \leq \mathcal{D}(O_1, O_2) \tag{7.3}$$

for every pair of objects O_1, O_2.

Proof: Let Q be the query object, O be a qualifying object, and ϵ be the tolerance. We want to prove that if the object O qualifies for the query, then it will be retrieved when we issue a range query on the feature space. That is, we want to prove that

$$\mathcal{D}(Q, O) \leq \epsilon \Rightarrow \mathcal{D}_{feature}(\mathcal{F}(Q), \mathcal{F}(O)) \leq \epsilon$$

However, this is obvious, since

$$\mathcal{D}_{feature}(\mathcal{F}(Q), \mathcal{F}(O)) \leq \mathcal{D}(Q, O) \leq \epsilon$$

Thus, the proof is complete. □

We have just proved that lower-bounding the distance works correctly for range queries. Will it work for the other queries of interest, like 'all-pairs' and 'nearest neighbor' ones? The answer is affirmative in both cases: An 'all-pairs' query can easily be handled by a 'spatial join' on the points of the feature space: using a similar reasoning as before, we see that the resulting set of pairs will be a superset of the qualifying pairs. For the nearest-neighbor query, the following algorithm guarantees no false dismissals: (a) find the point $\mathcal{F}(P)$ that is the nearest neighbor to the query point $\mathcal{F}(Q)$ (b) issue a range query, with query object Q and radius $\epsilon = \mathcal{D}(Q, P)$ (ie, the actual distance between the query object Q and data object P. For more details and for an application of this algorithm on tumor-like shapes, see [KSF+96].

In conclusion, the proposed generic approach to indexing multimedia objects for fast similarity searching shown in Figure 7.3 (named '*GEMINI*' for *GEneric Multimedia object INdexIng*):

The first two steps of GEMINI deserve some more discussion: The first step involves a domain expert. The methodology focuses on the *speed* of search only; the quality of the results is completely relying on the distance function that the expert will provide. Thus, GEMINI will return *exactly the same* response-set (and therefore, the same quality of output) with what the sequential scanning of the database would provide; the only difference is that GEMINI will be faster.

The second step of GEMINI requires intuition and imagination. It starts by trying to answer the question (referred to as the '*feature-extracting*' question for the rest of this work):

Algorithm 7.2 ('GEMINI') (GEneric Multimedia INdexIng approach):

1. determine the distance function $\mathcal{D}()$ between two objects
2. find one or more numerical feature-extraction functions, to provide a 'quick and dirty' test
3. prove that the distance in feature space *lower-bounds* the actual distance $\mathcal{D}()$, to guarantee correctness
4. use a SAM (eg., an R-tree), to store and retrieve the k-d feature vectors

Figure 7.3 Pseudo-code for the GEMINI algorithm.

'Feature-extracting' question: *If we are allowed to use only one numerical feature to describe each data object, what should this feature be?*

The successful answers to the above question should meet two goals: (a) they should facilitate step 3 (the distance lower-bounding) and (b) they should capture most of the characteristics of the objects.

We give case-studies of the GEMINI algorithm in the next two Chapters. The first involves 1-d time sequences, and the second focuses on 2-d color images. We shall see that the approach of the 'quick-and-dirty' filter, in conjunction with the lower-bounding lemma (Lemma 1), can lead to solutions to two problems:

- The dimensionality curse (time sequences).
- The 'cross-talk' of features (color images).

For each case study we (a) describe the objects and the distance function (b) show how to apply the lower-bounding lemma and (c) give experimental results, on real or realistic data. In Chapter 10 we show how to extend the idea of a 'quick-and-dirty' filter to handle sub-pattern matching in time sequences. In Chapter 11 we present '*FastMap*', an automated method of extracting features, for a given set of objects $\mathcal{O}$ and for a given distance function $\mathcal{D}()$.

8

1-D TIME SEQUENCES

Here the goal is to search a collection of (equal-length) time sequences, to find the ones that are similar to a desirable sequence. For example, '*in a collection of yearly stock-price movements, find the ones that are similar to IBM*'.

8.1 DISTANCE FUNCTION

According to GEMINI (Algorithm 7.2), the first step is to determine the distance measure between two time sequences. As in [AFS93], we chose the Euclidean distance (Eq. 7.2), because it is the distance of choice in financial and forecasting applications (e.g., [LeB92]). Fast indexing for additional, more elaborate distance functions that include time-warping [SK83] [WTK86] [RJ93] is the topic of ongoing research (eg., [GK95, JMM95]).

8.2 FEATURE EXTRACTION AND LOWER-BOUNDING

Having decided on the Euclidean distance as the dis-similarity measure, the next step of the GEMINI algorithm is to find some features that can lower-bound it. We would like a set of features that (a) preserve/lower-bound the distance and (b) carry much information about the corresponding time sequence, so that the false alarms are few. The second requirement suggests that we use 'good' features, that have much discriminatory power. In the stock-price example, a 'bad' feature would be, eg., the first-day's value: the reason is that

two stocks might have similar first-day values, yet they may differ significantly from then on. Conversely, two otherwise similar sequences, may agree everywhere, except for the first day's values. At the other extreme, we could use the values of *all* 365 days as features. However, although this would perfectly match the actual distance, it would lead to the 'dimensionality curse' problem.

Clearly, we need some better features. Applying the second step of the GEMINI algorithm, we ask the 'feature-extracting' question: *if we are allowed to use only one feature from each sequence, what would this feature be?* A natural answer is the average. By the same token, additional features could be the average of the first half, of the second half, of the first quarter, etc. Or, in a more systematic way, we could use the coefficients of the Fourier transform, and, for our case, the Discrete Fourier Transform (DFT) (see, eg., [OS75], or Appendix B). For a signal $\vec{x} = [x_i]$, $i = 0, \ldots, n-1$, let X_f denote the n-point DFT coefficient at the f-th frequency ($f = 0, \ldots, n-1$). Also, let $\vec{X} = [X_f]$ be the n-point DFT transform of $\vec{x}$. Appendix B provides a quick introduction to the basic concepts of the DFT.

The third step of the GEMINI methodology is to show that the distance in feature space lower-bounds the actual distance. The solution is provided by Parseval's theorem [OS75], which states that the DFT preserves the energy of a signal, as well as the distances between two signals:

$$\mathcal{D}(\vec{x}, \vec{y}) = \mathcal{D}(\vec{X}, \vec{Y}) \tag{8.1}$$

where $\vec{X}$ and $\vec{Y}$ are Fourier transforms of $\vec{x}$ and $\vec{y}$ respectively.

Thus, if we keep the first $k(\leq n)$ coefficients of the DFT as the features, we lower-bound the actual distance:

$$\begin{aligned} \mathcal{D}_{feature}(\mathcal{F}(\vec{x}), \mathcal{F}(\vec{y})) &= \sum_{f=0}^{k-1} |X_f - Y_f|^2 \\ &\leq \sum_{f=0}^{n-1} |X_f - Y_f|^2 \;=\; \sum_{i=0}^{n-1} |x_i - y_i|^2 \;\equiv\; \mathcal{D}(\vec{x}, \vec{y}) \end{aligned}$$

because we ignore positive terms from the above Equation. Thus, there will be *no false dismissals*, according to Lemma 1.

Notice that our GEMINI approach can be applied with *any* orthonormal transform, such as, the Discrete Cosine transform (DCT) (see [Wal91] or Appendix B.4), the wavelet transform (see [RBC+92], [PTVF92], or Appendix C) etc., because

they all preserve the distance between the original and the transformed space. In fact, our response time will improve with the ability of the transform to concentrate the *energy*: the fewer the coefficients that contain most of the energy, the more accurate our estimate for the actual distance, the fewer the false alarms, and the faster our response time. The energy of a signal is the sum of squares of its elements (see Definition A.7 in the Appendix). Thus, the performance results presented next are just pessimistic bounds; better transforms will achieve even better response times.

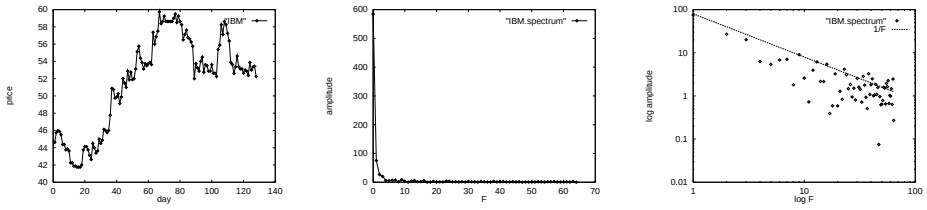
In addition to being readily available, (eg., in mathematical symbolic manipulation packages, like 'Mathematica', 'S', 'maple' etc.), the DFT concentrates the energy in the first few coefficients, for a large class of signals, the *colored noises*. These signals have a skewed energy spectrum that drops as $O(f^{-b})$. The *energy spectrum* or *power spectrum* of a signal is the square of the amplitude $|X_f|$, as a function of the the frequency f (see Appendix B).

- For b=2, we have the so-called *random walks* or *brown noise*, which model successfully stock movements and exchange rates (e.g., [Man77]). Our mathematical argument for keeping the first few Fourier coefficients agrees with the intuitive argument of the Dow Jones theory for stock price movement (see, for example, [EM66]). This theory tries to detect *primary* and *secondary* trends in the stock market movement, and ignores *minor* trends. Primary trends are defined as changes that are larger than 20%, typically lasting more than a year; secondary trends show 1/3-2/3 relative change over primary trends, with a typical duration of a few months; minor trends last roughly a week. From the above definitions, we conclude that primary and secondary trends correspond to strong, low frequency signals while minor trends correspond to weak, high frequency signals. Thus, the primary and secondary trends are exactly the ones that our method will automatically choose for indexing.

- With even more skewed spectrum ($b > 2$), we have the *black noises* [Sch91]. Such signals model successfully, for example, the water level of rivers and the rainfall patterns as they vary over time [Man77].

- with b=1, we have the *pink noise*. Birkhoff's theory [Sch91] claims that 'interesting' signals, such as musical scores and other works of art, consist of 'pink noise', whose energy spectrum follows $O(f^{-1})$. The argument of the theory is that white noise with $O(f^0)$ energy spectrum is completely unpredictable, while brown noise with $O(f^{-2})$ energy spectrum is too predictable and therefore 'boring'. The energy spectrum of pink noise lies in-between.

As an illustration of a 'colored noise', Figure 8.1 plots the closing prices of the IBM stock, 8/30/93 - 4/20/94, along with the amplitude spectrum in linear and logarithmic scales. Notice how skewed the spectrum is, as well as how closely it is approximated by the theoretically expected $1/f$ line.



(a) IBM stock (b) spectrum (linear scales) (c) spectrum (log scales)

Figure 8.1 (a) Closing prices of IBM stock, 8/30/93 - 4/20/94, (b) its DFT amplitude spectrum in linear scales and (c) in logarithmic scales, along with the theoretically expected $1/f$ line.

Additional financial data sets with similar behavior are available from `ftp://sfi.santafe.edu/pub/Time-Series/competition` (Dataset 'C', with the Swiss franc - U.S. dollar exchange rate), or from `http://www.ai.mit.edu/stocks.html`, where there are stock prices for several companies.

In addition to 1-d signals (stock-price movements and exchange rates), it is believed that several families of higher dimensionality signals belong to the family of 'colored noises', with skewed spectrum. For example, 2-d signals, like photographs, typically exhibit a few strong coefficients in the lower spatial frequencies. The JPEG image compression standard [Wal91] exactly exploits this phenomenon, effectively ignoring the high-frequency components of the Discrete Cosine Transform, which is closely related to the Fourier transform (see Appendix B.4).

8.3 EXPERIMENTS

The above indexing method was implemented and compared against sequential scanning, on a collection of synthetic random walks. See [AFS93] for more details. As expected, the sequential scanning was outperformed.

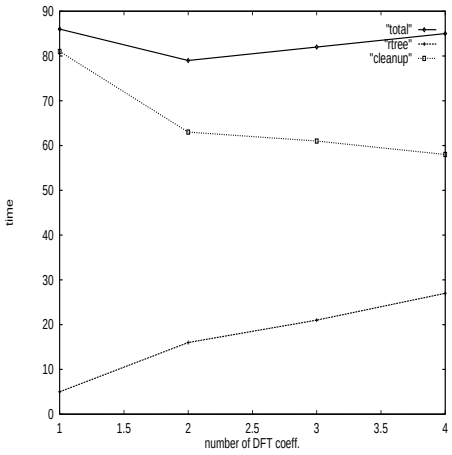


Figure 8.2 Breakup of the execution time, for range queries: response time versus number of DFT coefficients.

An interesting point is how to determine the number k of DFT coefficients to retain. Figure 8.2 shows the break-up of the response time versus the number k of DFT coefficients retained. The diamonds, squares and crosses indicate total time, post-processing time and R-tree time, respectively. Notice that, as we keep more coefficients, the R-tree becomes bigger and slower, but more accurate (fewer false alarms, and therefore shorter post-processing time). This trade-off reaches an equilibrium for k=2 or 3.

The major conclusions from the application of the GEMINI method on time sequences are the following:

1. GEMINI can be successfully applied to time sequences, and specifically to the ones that behave like 'colored noises' (stock prices movements, currency exchange rates, water-level in rivers etc.)

2. For signals with skewed spectrum like the above ones, the minimum in the response time is achieved for a small number of Fourier coefficients ($k = 1-3$). Moreover, the minimum is rather flat, which implies that a sub-optimal choice for k will give search time that is close to the minimum. Thus, with the help of the lower-bounding lemma and the energy-concentrating properties of the DFT, we managed to avoid the 'dimensionality curse'.

3. The success in 1-d sequences suggests that the proposed GEMINI method is promising for 2-d or higher-dimensionality signals, if those signals also have skewed spectrum. The success of JPEG (that uses DCT) indicates that real images indeed have a skewed spectrum.

Exercises

Exercise 8.1 [10] *Write a program to generate a random walk. Let each step be the output of a fair coin tossing: +1 or -1, with probability 50% each.*

Exercise 8.2 [10] *Compute and plot the spectrum of the above random walk, using some existing DFT package; also plot the 1/f line.*

Exercise 8.3 [15] *Use some time sequences from, eg., [Ton90, BJR94], and compute their DFT spectrum. List your observations.*

Exercise 8.4 [25] *Experiment with the 'energy concentrating' properties of DFT versus the DCT, for the sequences of the previous exercise: For each sequence, keep the k strongest coefficients of the DFT and the DCT transform; plot the squared error (sum of squares of omitted coefficients), as a function of k, for each of the two transforms. List your observations.*

Exercise 8.5 [20] *Write a program that will generate 'pink noise', that is, a signal with 1/f amplitude spectrum. (Hint: use the random walk above from Exercise 8.1, compute its DFT spectrum, divide each Fourier coefficient appropriately, and invert).*

9

2-D COLOR IMAGES

Retrieving images by content attracts high and increasing interest [JN92, OS95, FSN+95]. Queries on content may focus on color, texture, shape, position, etc. Potential applications include medical image databases ('*Give me other images that contain a tumor with a texture like this one*'), photo-journalism ('*Find images that have blue at the top and red at the bottom*'), art, fashion, cataloging, retailing etc..

In this chapter we present a color indexing algorithm and specifically the distance functions and the application of the GEMINI approach. We focus on 'whole-match' queries, or, equivalently, 'queries by example', where the user chooses an existing image (or draws one with a sketch-pad) and asks for similar color images. This color indexing algorithm was developed within the QBIC system of IBM [FBF+94]; see [NBE+93, Equ93, FSN+95] for more details on the shape and texture indexing algorithms of QBIC and their precision-recall performance evaluation.

Past work on image retrieval has resulted in many systems and methods to match images according to color, shape, texture and relative position. For color, a typical choice is the color histograms which we describe next; for shape-matching, the turning angle [Hor86], the moments of inertia [FBF+94] or the pattern spectrum [KSF+96] are among the choices; for texture, the directionality, granularity and contrast [Equ93] are a good set of features; for the relative position, the 2-D strings method and its variants have been used [CSY87, PO95]. However, although there is a lot of work on image matching in the Machine Vision community and on fast searching in the database community, the former typically focuses on the quality of the matching, while the latter focuses on the speed of the search; it is only recently that the two

communities have started collaborating [JN92, NBE+93], in an attempt to provide fast *and* effective image retrieval by content.

Recent surveys on image matching are in [FBF+94, PF96]. A presentation of image retrieval systems is in the special issue of the IEEE Computer (Sept. 1995) [GR95].

9.1 DISTANCE FUNCTION

We mainly focus on the color features, because color presents an interesting problem (namely, the '*cross-talk*' of features), which can be resolved by the proposed 'GEMINI' approach (algorithm 7.2). Features for shape and texture are described in [FSN+95], and can be easily mapped into k-d points.

Each object is a color image, that is, a 2-d array of pixels; every pixel has 3 color components (eg., 1 byte for 'red', 1 byte for 'green' and 1 byte for 'blue'). For each image, we compute an h-element color histogram using h colors. Conceptually, h can be as high as 2^{24} colors, with each color being denoted by a point in a 3-dimensional color space. In practice, we cluster similar colors together using an agglomerative clustering technique [DH73a], and choose one representative color for each bucket (= 'color bin'). Typical numbers of color bins are $h = 256$ and $h = 64$. Each component in the color histogram is the percentage of pixels that are most similar to that color. Figure 9.1 gives an example of such a histogram of a fictitious photograph of a sunset: there are many red, pink, orange and purple pixels, but only few white and green ones.

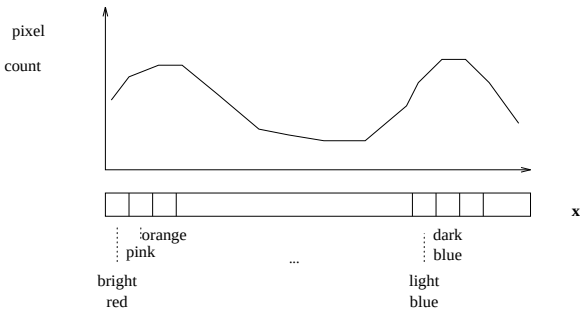


Figure 9.1 An example of a color histogram of a fictitious sunset photograph: Many red, pink, orange, purple and blue-ish pixels; few yellow, white and green-ish ones

Once these histograms are computed, one method to measure the distance between two histograms ($h \times 1$ vectors) $\vec{x}$ and $\vec{y}$ is given by

$$d^2_{hist}(\vec{x}, \vec{y}) = (\vec{x} - \vec{y})^t \times \mathbf{A} \times (\vec{x} - \vec{y}) = \sum_i^h \sum_j^h a_{ij}(x_i - y_i)(x_j - y_j) \qquad (9.1)$$

where the superscript 't' indicates matrix transposition, '$\times$' indicates matrix multiplication, and the color-to-color similarity matrix $\mathbf{A}$ has entries a_{ij} which describe the similarity between color i and color j, with $a_{ii} = 1$ for every i.

9.2 LOWER-BOUNDING

If we try to use the color-histograms as feature vectors in the GEMINI approach, there are two obstacles: (a) The *'dimensionality curse'* (h may be large, e.g. 64 or 256 for color features) and, most importantly, (b) the *quadratic nature of the distance function*: The distance function in the feature space involves 'cross-talk' among the features (see Eq. 9.1), and it is thus a full quadratic form involving all cross terms. Not only is such a function much more expensive to compute than the Euclidean and any L_p distance, but it also precludes the use of spatial access methods ('SAMs'), because SAMs implicitly assume that there is no cross-talk. Figure 9.2 illustrates the situation: to compute the distance between the two color histograms $\vec{x}$ and $\vec{q}$, the, eg., bright-red component of $\vec{x}$ has to be compared not only to the bright-red component of $\vec{q}$, but also to the pink, orange etc. components of $\vec{q}$.

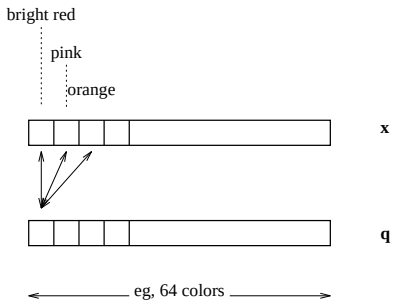


Figure 9.2 Illustration of the 'cross-talk' between two color histograms

To resolve the cross-talk problem, we resort to the '*GEMINI*' approach (algorithm 7.2). The first step of the algorithm has been done: the distance function

between two color images is given by Eq. 9.1: $\mathcal{D}() = d_{hist}()$. The second step is to find one or more numerical features, whose Euclidean distance would lower-bound $d_{hist}()$. Thus, we ask the 'feature-extracting' question again: *If we are allowed to use only one numerical feature to describe each color image, what should this feature be?* According to the previous chapter on time sequences, we can consider some average value, or the first few coefficients of the 2-dimensional DFT transform. Since we have three color components, (eg., Red, Green and Blue), we could consider the average amount of red, green and blue in a given color image.

Notice that different color spaces can be used, with absolutely no change in the indexing algorithms. Thus, we continue the discussion with the RGB color space. This means that the color of an individual pixel is described by the triplet (R,G,B) (for 'R'ed, 'G'reen, 'B'lue). The average color of an image $\bar{x} = (R_{avg}, G_{avg}, B_{avg})^t$, is defined in the obvious way, with

$$\begin{aligned} R_{avg} &= (1/P)\sum_{p=1}^{P} R(p) \\ G_{avg} &= (1/P)\sum_{p=1}^{P} G(p) \\ B_{avg} &= (1/P)\sum_{p=1}^{P} B(p) \end{aligned}$$

where P is the number of pixels in the image, and $R(p)$, $G(p)$, and $B(p)$ are the red, green and blue components (intensities, typically in the range 0-255) respectively of the p-th pixel. Given the average color vectors $\bar{x}$ and $\bar{y}$ of two images, we define $d_{avg}()$ as the Euclidean distance between the 3-dimensional average color vectors,

$$d^2_{avg}(\bar{x}, \bar{y}) = (\bar{x} - \bar{y})^t \times (\bar{x} - \bar{y}) \;=\; \sum_{i=1}^{3} (x_i - y_i)^2 \qquad (9.2)$$

The third step of the GEMINI algorithm is to prove that our feature distance $\mathcal{D}_{feature}() \equiv d_{avg}()$ lower-bounds the actual distance $\mathcal{D}() \equiv d_{hist}()$. Indeed, this is true, as an application of the so-called *QDB*- 'Quadratic Distance Bounding' Theorem [FBF+94].

The result is that, given a color query, our retrieval proceeds by first filtering the set of images based on their average (R, G, B) color, then doing a final,

more accurate matching using their full h-element histogram. The resulting speedup is discussed next.

9.3 EXPERIMENTS

Experiments are reported in [FBF+94], on a database of N=924 color image histograms, each of h=256 colors, of assorted natural images. The proposed method requires from a fraction of a second up to $\approx$ 4 seconds, while sequential scanning with the color-histogram distance (Eq. 9.1) requires $\approx$ 10 seconds. The performance gap is expected to increase for larger databases.

Thus, the conclusions are the following:

- The 'GEMINI' approach (ie., the idea to extract some features for a quick-and-dirty test) motivated a fast method, using the average RGB distance ($d_{avg}()$); it also motivated a strong theorem (the so-called *QDB*- 'Quadratic Distance Bounding' Theorem [FBF+94]) which guarantees the correctness in our case.
- In addition to resolving the cross-talk problem, 'GEMINI' solved the 'dimensionality curse' problem at no extra cost, requiring only k=3 features, as opposed to h=64 or 256 that $d_{hist}()$ required.

10

SUB-PATTERN MATCHING

10.1 INTRODUCTION

Up to now, we have examined the 'whole-match' case. The goal in this chapter is to extend the 'GEMINI' approach of the 'quick-and-dirty' test, so that we can handle sub-pattern matching queries. We focus on 1-d time series, to illustrate the problem and the solution more clearly. Then, the problem is defined as follows:

- We are given a collection of N sequences of real numbers S_1, S_2, S_N, each one of potentially different length.
- The user specifies query subsequence Q of length $Len(Q)$ (which may vary) and the tolerance ϵ, that is, the maximum acceptable dis-similarity (= distance).
- We want to find quickly all the sequences S_i ($1 \leq i \leq N$), along with the correct offsets p, such that the subsequence $S_i[p : p + Len(Q) - 1]$ matches the query sequence: $\mathcal{D}(Q, S_i[p : p + Len(Q) - 1]) \leq \epsilon$.

As in Chapter 8, we use the Euclidean distance as the dis-similarity measure $\mathcal{D}()$. The brute-force solution is to examine sequentially every possible subsequence of the data sequences for a match. We shall refer to this method by *'SequentialScan'* method. Next, we describe a method that uses a small space overhead, to achieve up to 2 orders of magnitudes savings over the *'SequentialScan'* method [FRM94].

10.2 SKETCH OF THE APPROACH - *'ST-index'*

Without loss of generality, we assume that the minimum query length is w, where w (≥ 1) depends on the application. For example, in stock price databases, analysts are interested in weekly or monthly patterns because shorter patterns are susceptible to noise [EM66]. Notice that we never lose the ability to answer shorter than w queries, because we can always resort to sequential scanning.

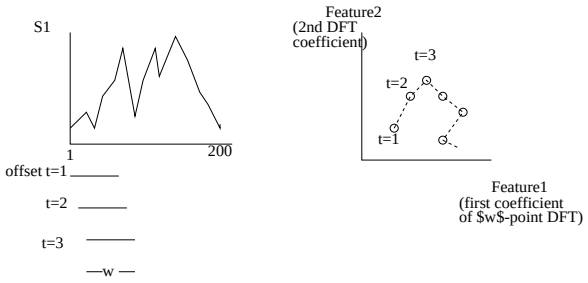


Figure 10.1 Illustration of the way that trails are created in feature space

Generalizing the reasoning of the method for 'whole matching', we use a *sliding window* of size w and place it at every possible position (offset), on every data sequence. For each such placement of the window, we extract the features of the subsequence inside the window. Thus, a data sequence of length $Len(S)$ is mapped to a trail in feature space, consisting of ($Len(S)$ -w+1) points: one point for each possible offset of the sliding window. Figure 10.1 gives an example of a trail: Consider the sequence S_1, and assume that we keep the first k=2 features (eg, the amplitude of the first and second coefficient of the w-point DFT). When the window of length w is placed at offset=0 on S_1, we obtain the first point of the trail; as the window slides over S_1, we obtain the rest of the points of the trail.

The straightforward way to index these trails would be to keep track of the individual points of each trail, storing them in a spatial access method. We call this method *'I-naive'* method, where 'I' stands for 'Index' (as opposed to sequential scanning). However, storing the individual points of the trail in an R-tree is inefficient, both in terms of space as well as search speed. The reason is that, almost every point in a data sequence will correspond to a point in the k-dimensional feature space, leading to an index with a 1:k increase in storage requirements. Moreover, the search performance will also suffer because the R-tree will become tall and slow. As shown in [FRM94], the *'I-naive'* method

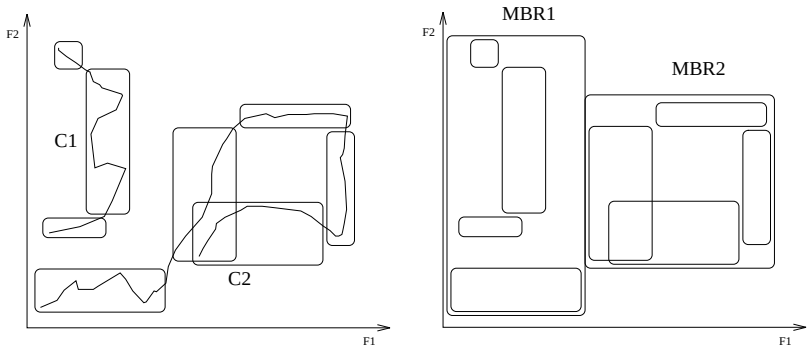


Figure 10.2 Example of (a) dividing trails into sub-trails and MBRs, and (b) grouping of MBRs in larger ones.

ended up being almost twice as slow as the *'SequentialScan'*! Thus, we want to improve the *'I-naive'* method, by making the representation of the trails more compact.

Here is where the idea of a 'quick-and-dirty' test leads to a solution: Instead of laboriously keeping track of each and every point of a trail in feature space, we propose to exploit the fact that successive points of the trail will probably be similar, because the contents of the sliding window in nearby offsets will be similar. We propose to divide the trail of a given data sequence into sub-trails and represent each of them with its *minimum bounding (hyper)-rectangle (MBR)*. Thus, instead of storing thousands of points of a given trail, we shall store only a few MBRs. More importantly, at the same time we still guarantee 'no false dismissals': when a query arrives, we shall retrieve all the MBRs that intersect the query region; thus, we shall retrieve all the qualifying sub-trails, plus some false alarms (sub-trails that do not intersect the query region, while their MBR does).

Figure 10.2(a) gives an illustration of the proposed approach. Two trails are drawn; the first curve, labeled $C1$ (in the north-west side), has been divided into three sub-trails (and MBRs), whereas the second one, labeled $C2$ (in the south-east side), has been divided in five sub-trails. Notice that it is possible that MBRs belonging to the same trail may overlap, as $C2$ illustrates.

Thus, the idea is to map a data sequence into a *set of rectangles* in feature space. This yields significant improvements with respect to space, as well as

with respect to response time, as we shall see in section 10.3. Each MBR corresponds to a whole sub-trail, that is, points in feature space that correspond to successive positionings of the sliding window on the data sequences.

These MBRs can be subsequently stored in a spatial access method, such as an R-tree (see Chapter 5). Figure 10.2(b) shows how the eight leaf-level MBRs of Figure 10.2(a) will be grouped to form two MBRs at the next higher level, assuming a fanout of 4 (i.e. at most 4 items per non-leaf node). Note that the higher-level MBRs may contain leaf-level MBRs from different data sequences. For example, in Figure 10.2(b) notice that the left-side MBR1 contains a part of the C2 curve.

This completes the sketch of the proposed index structure. We shall refer to it by *'ST-index'* , for 'Sub-Trail index'. There are two questions that we have to answer, to complete the description of the method:

- Insertions: When a new data sequence is inserted, what is a good way to divide its trail in feature space into sub-trails? The idea [FRM94] is to use an adaptive, heuristic algorithm, which will break the trail into sub-trails, so that the resulting MBRs of the sub-trails have small volume (and, therefore, result in few disk accesses on search).
- Queries: How to handle queries, and especially the ones that are longer than w? Figure 10.3 shows how to handle queries of length w: the query sub-sequence is translated into a point Q in feature space; all the MBRs that are within radius ϵ from Q are retrieved. Searching for longer queries is handled by breaking the query pattern into pieces of length w, searching the R-tree for each piece, and then merging the results. Figure 10.4 illustrates the algorithm: the query sequence is broken into p=2 pieces, each of length w; each piece gives rise to a range query in feature space.

More details and the correctness proofs for the above algorithms are in [FRM94].

10.3 EXPERIMENTS

The above method was implemented and tested on real stock-price time sequences totaling 329,000 points [FRM94]. The proposed method achieved up to 100 times better response time, compared to the sequential scanning.

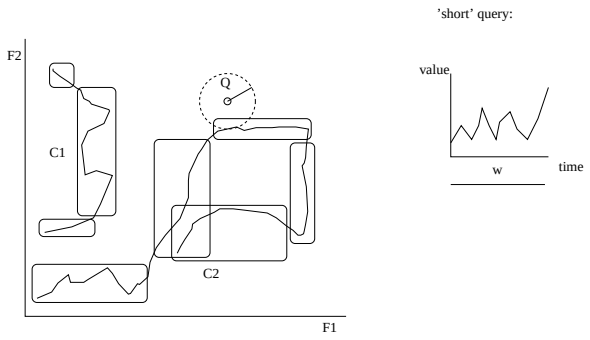


Figure 10.3 Illustration of the search algorithm for minimum-length queries. The query becomes a sphere of radius ϵ in feature space.

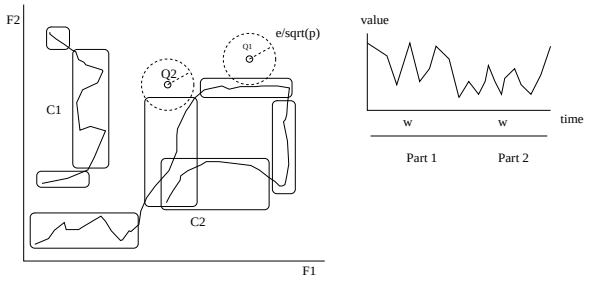


Figure 10.4 Illustration of the search algorithm for a longer query. The query is divided in p=2 pieces of length w each, giving rise to p range queries in feature space

The conclusion is that the idea of using a 'quick-and-dirty' filter pays off again. Every sequence is represented *coarsely* by a set of MBRs in feature space; despite the loss of information, these MBRs provide the basis for quick filtering, which eventually achieves large savings over the sequential scanning.

Exercises

Exercise 10.1 [40] *Implement a system which can search a collection of stock-prices for user-specified patterns. Assume that the minimum length is one week; Use artificially generated random walks (exercise 8.1), or real data, from* `http://www.ai.mit.edu/ stocks.html`.

11

FASTMAP

11.1 INTRODUCTION

In the previous chapters we saw the 'GEMINI' approach, which suggests that we rely on domain experts to derive k feature-extraction functions, thus mapping each object into a point in k-dimensional space. Then the problem is reduced to storing, retrieving and displaying k-dimensional points, for which there is a plethora of algorithms available.

However, it is not always easy to derive the above feature-extraction functions. Consider the case, eg., of typed English words, where the distance function is the editing distance (minimum number of insertion, deletions and substitutions to transform one string to the other). It is not clear which the features should be in this case. Similarly, in matching digitized voice excerpts, we typically have to do some time-warping [SK83], which makes it difficult to design feature-extraction functions.

Overcoming these difficulties is exactly the motivation behind this chapter. Automatically mapping the objects into points in some k-d space provides two major benefits:

1. It can accelerate the search time for queries. The reason is that we can employ highly fine-tuned Spatial Access Methods (SAMs), like the R^*-trees [BKSS90] and the z-ordering [Ore86]. These methods provide fast searching for range queries as well as spatial joins [BKSS94].
2. It can help with visualization, clustering and data-mining: Plotting objects as points in k=2 or 3 dimensions can reveal much of the structure of

the dataset, such as the existence of major clusters, the general shape of the distribution (linear versus curvilinear versus Gaussian) etc.. These observations can provide powerful insights in formulating hypotheses and discovering rules.

Next, we shall use the following terminology:

Definition 11.1 The k-dimensional point P_i that corresponds to the object O_i, will be called 'the *image*' of object O_i. That is, $P_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$

Definition 11.2 The k-dimensional space containing the 'images' will be called *target space*.

Given the above, the problem is defined as follows (see Figure 11.1 for an illustration):

Problem

Given N objects and distance information about them (eg., an $N \times N$ distance matrix, or simply the distance function $\mathcal{D}(*,*)$ between two objects)
find N points in a k-dimensional space,
such that the distances are maintained as well as possible.

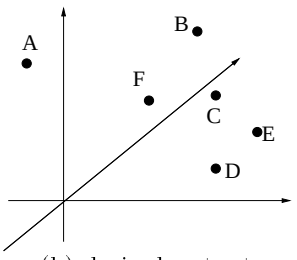
A special case is the situation where the N objects are n-d vectors. Then, goal is to do *dimensionality reduction*, mapping them into points in k-d space, preserving the (Euclidean) distances as well as possible. In this case, the optimal solution is given by the *Karhunen-Loeve* ('K-L') transform, which is described in detail in Appendix D.1.

For the general case, we expect that the distance function $\mathcal{D}()$ is non-negative, symmetric and obeys the triangular inequality. In the 'target' (k-d) space, we typically use the Euclidean distance, because it is invariant under rotations. Alternative distance metrics could be any of the L_p metrics, like the L_1 ('city-block' or 'Manhattan' distance).

The ideal mapping should fulfill the following requirements:

	A	B	C	D	E	F
A	0	8	7	7	9	5
B	-	0	2	4	4	2
C	-	-	0	1.5	1	2
D	-	-	-	0	1	3
E	-	-	-	-	0	3
F	-	-	-	-	-	0

(a) input



(b) desired output

Figure 11.1 (a) Six objects and their pair-wise distance information; (b) 3-d points that try to approximate the given distance information.

1. It should be fast to compute: $O(N)$ or $O(N \log N)$, but not $O(N^2)$ or higher, because the cost will be prohibitive for large databases.
2. It should preserve distances, leading to small discrepancies (low '*stress*' - see (Eq. 11.1)).
3. It should provide a very fast algorithm to map a new object (eg., a query object) to its image. The algorithm should be $O(1)$ or $O(\log N)$. This requirement is vital for 'queries-by-example'.

The outline of this chapter is as follows. In section 11.2 we present a brief survey of Multi-Dimensional Scaling (MDS). In section 11.3 we describe 'FastMap', a linear, approximate solution [FL95]. In section 11.4 we show the results of FastMap on some real data (document vectors). Finally, section 11.5 lists the conclusions.

11.2 MULTI-DIMENSIONAL SCALING (MDS)

Multi-dimensional scaling (MDS) is used to discover the underlying (spatial) structure of a set of data items from the (dis)similarity information among them. There are several variations, but the basic method (eg., see [KW78]) is described next. The method expects (a) a set of N items, (b) their pair-wise (dis)similarities and (c) the desirable dimensionality k. Then, the algorithm

will map each object to a point in a k dimensional space, to minimize the *stress* function:

$$stress = \sqrt{\frac{\sum_{i,j} \left(\hat{d}_{ij} - d_{ij}\right)^2}{\sum_{i,j} {d_{ij}}^2}} \tag{11.1}$$

where d_{ij} is the dissimilarity measure between object O_i and object O_j and $\hat{d}_{ij}$ is the (Euclidean) distance between their 'images': points P_i and P_j. The 'stress' function gives the relative error that the distances in k-d space suffer from, on the average.

To achieve its goal, MDS starts with a guess and iteratively improves it, until no further improvement is possible. In its simplest version, the algorithm works roughly as follows: It originally assigns each item to a k-d point (eg., using some heuristic, or even at random). Then, it examines every point, computes the distances from the other $N-1$ points and moves the point to minimize the discrepancy between the actual dissimilarities and the estimated k-d distances. Technically, MDS employs the method of 'steepest descent' to update the positions of the k-d points. Intuitively, it treats each pair-wise distance as a 'spring' between the two points; then, the algorithm tries to re-arrange the positions of the k-d points to minimize the 'stress' of the springs.

MDS has been used in numerous, diverse applications, to help visualize a set of objects, given their pair-wise similarities. However, for our applications, MDS suffers from two drawbacks:

- It requires $O(N^2)$ time, where N is the number of items. Thus, it is impractical for large datasets. In the applications that MDS has been used, the number of items was small (typically, N=10-100).
- Its use for fast retrieval is questionable: In the 'query-by-example' setting, the query item has to be mapped to a point in k-d space. MDS is not prepared for this operation: Given that the MDS algorithm is $O(N^2)$, an incremental algorithm to search/add a new item in the database would be $O(N)$ at best. Thus, the complexity of answering a query would be as bad as sequential scanning.

The above two drawbacks are the motivation behind the next section.

Symbols	Definitions.
N	Number of objects in database
k	dimensionality of 'target space'
$\mathcal{D}(*,*)$	the distance function between two objects
$\| \vec{x} \|$	the length (= L_2 norm) of vector $\vec{x}$
(AB)	the length of the line segment AB

Table 11.1 Summary of Symbols and Definitions

11.3 A FAST, APPROXIMATE ALTERNATIVE: FASTMAP

The goal is to find N points in k-d space, whose Euclidean distances will match the distances of a given $N \times N$ distance matrix. Table 11.1 lists the symbols and their definitions. The key idea is to pretend that objects are indeed points in some unknown, n-dimensional space, and to try to project these points on k mutually orthogonal directions. The challenge is to compute these projections from the distance matrix only, since it is the only input we have. For the rest of this discussion, an object will be treated as if it were a point in an n-d space, with unknown n.

The heart of the 'FastMap' method is to project the objects on a carefully selected 'line'. To do that, we choose two objects O_A and O_B (referred to as *'pivot objects'* from now on), and consider the 'line' that passes through them in n-d space. The algorithm to choose pivot objects uses a linear, approximate heuristic, and is discussed later (see Figure 11.4).

The projections of the objects on that line are computed using the *cosine law*:

$$ {d_{b,i}}^2 = {d_{a,i}}^2 + {d_{a,b}}^2 - 2x_i d_{a,b} \tag{11.2} $$

See Figure 11.2 for an illustration. Eq. 11.2 can be solved for x_i, the first coordinate of object O_i:

$$ x_i = \frac{{d_{a,i}}^2 + {d_{a,b}}^2 - {d_{b,i}}^2}{2d_{a,b}} \tag{11.3} $$

In the above equations, d_{ij} is a shorthand for the distance $\mathcal{D}(O_i, O_j)$ (for $i, j = 1, \ldots, N$). Notice that the computation of x_i only needs the distances between objects, which are given.

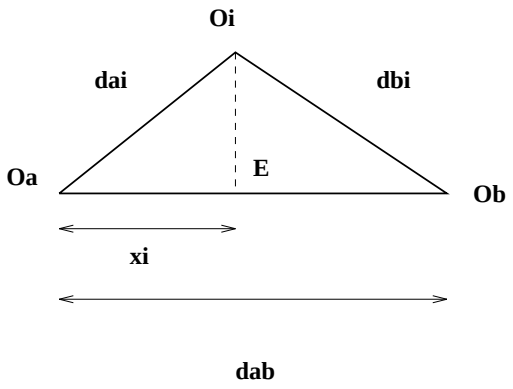


Figure 11.2 Illustration of the 'cosine law' - projection on the line $O_A O_B$.

Observe that, thanks to Eq. 11.3, we can map objects into points on a line, preserving some of the distance information: For example, if O_i is close to the pivot O_A, then x_i will be small. Thus, we have the solution to our original problem, for k=1.

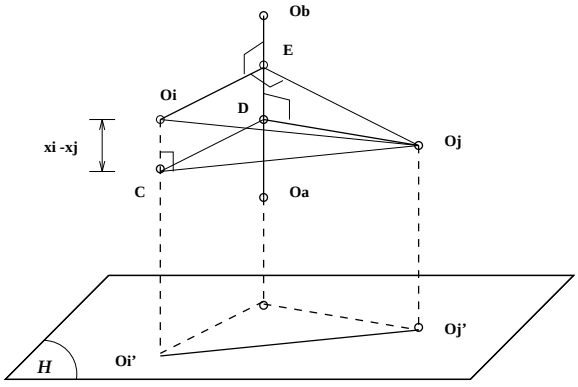


Figure 11.3 Projection on a hyper-plane $\mathcal{H}$, perpendicular to the line $O_A O_B$ of the previous figure.

To extend this method for 2-d and k-d target spaces, we keep on pretending that the objects are indeed points in n-d space: We consider a $(n-1)$-d hyper-plane $\mathcal{H}$ that is perpendicular to the line (O_A, O_B); then, project our objects on this hyper-plane. Let $O_i{}'$ stand for the projection of O_i (for $i = 1, \ldots, N$).

Algorithm 11.1 FastMap (k, $\mathcal{D}()$, $\mathcal{O}$)

k: number of dimensions;
$\mathcal{D}()$: the distance function;
$\mathcal{O}$: the set of objects.

1. Pick-pivots O_A and O_B from $\mathcal{O}$.
2. Project all objects O_i along the line O_A - O_B.
3. Call FastMap($k-1$, $\mathcal{D}'()$, $\mathcal{O}$), unless $k = 0$.

Figure 11.4 Pseudo-code for the FastMap algorithm.

The problem is the same as the original problem, with n and k decreased by one.

The only missing part is to determine the distance function $\mathcal{D}'()$ between two of the projections on the hyper-plane $\mathcal{H}$, such as, ${O_i}'$ and ${O_j}'$. Once this is done, we can recursively apply the previous steps. Figure 11.3 depicts two objects O_i, O_j, and their projections ${O_i}'$, ${O_j}'$ on the $\mathcal{H}$ hyper-plane.

The distance function $\mathcal{D}'()$ is given by:

$$(\mathcal{D}'({O_i}', {O_j}'))^2 = (\mathcal{D}(O_i, O_j))^2 - (x_i - x_j)^2 \quad i, j = 1, \ldots, N \tag{11.4}$$

using the Pythagorean theorem on the triangle $O_i C O_j$ (see Figure 11.3).

Ability to compute the distance $\mathcal{D}'()$ allows us to project on a second line, lying on the hyper-plane $\mathcal{H}$, and, therefore, orthogonal to the first line (O_A, O_B) by construction. Thus, we can solve the problem for a 2-d 'target' space. More importantly, we can apply the same steps recursively, k times, thus *solving the problem for any k*.

Figure 11.4 gives the pseudo-code for FastMap.

To pick the pivot objects, we use a linear-time heuristic, to choose a pair of objects that are far-away from each other. Notice that finding the exact solution requires a quadratic number of steps ($\mathrm{O}(N^2)$), because we have to examine every possible pair at least once.

Algorithm 11.2 Pick-pivots ($\mathcal{O}$, $\mathcal{D}()$)

1. Chose arbitrarily an object; let it be the second pivot object O_B
2. Set O_A = (the object that is farthest apart from O_B)
3. Set O_B = (the object that is farthest apart from O_A)
4. Repeat the last two steps a fixed number of times
5. Report the objects O_A and O_B as the pivot objects

Figure 11.5 Heuristic to choose two distant objects for pivots.

11.4 CASE STUDY: DOCUMENT VECTORS AND INFORMATION RETRIEVAL.

Here we use the algorithm on an information retrieval application [SM83]. As we mentioned in section 6.5, in the vector space model, documents are represented as vectors in V-dimensional space, where V is the size of the vocabulary of the collection. For the English language, we can expect V to range from 2,000 up to and exceeding 100,000 (the vocabulary of every-day English, and the size of a very detailed dictionary, respectively [Pet80]). The coordinates of such vectors are called *term weights* and can be binary ('1' if the term appears in the document; '0' if not) or real-valued, with values increasing with the importance (eg., occurrence frequency) of the term in the document.

Consider two documents d_1 and d_2, with vectors $\vec{u_1}$, $\vec{u_2}$ respectively. As we mentioned, the similarity between two documents is typically measured by the *cosine similarity* of their vectors [SM83]:

$$similarity(d_1, d_2) = \frac{\vec{u_1} \circ \vec{u_2}}{\| \vec{u_1} \| \| \vec{u_2} \|}$$

where '$\circ$' stands for the inner product of two vectors and $\| * \|$ stands for the length (=Euclidean norm) of the vector. Clearly the cosine similarity takes values between -1 and 1. Figure 11.6 gives an example. There, $\cos(\theta)$ is considered as the similarity of the two vectors $\vec{u_1}$ and $\vec{u_2}$. Intuitively, the cosine similarity projects all the document vectors on the unit hyper-sphere (see vectors $\vec{u_{1,0}}$ and $\vec{u_{2,0}}$ in the Figure) and measures the cosine of the angle of the projections.

In order to apply FastMap, we first need to define a distance function that decreases with increasing similarity. From Figure 11.6 it would make sense

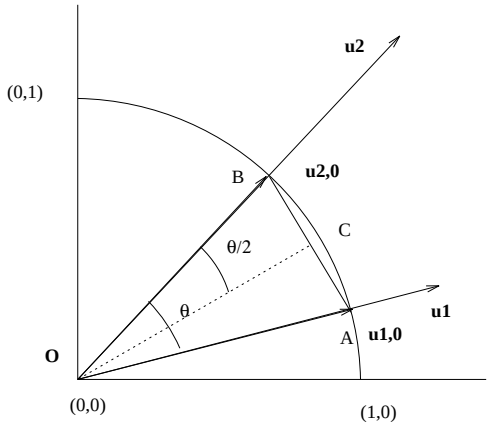


Figure 11.6 Two vectors $\vec{u_1}$, $\vec{u_2}$, their angle θ and the cosine similarity function $\cos(\theta)$

to use the length of the line segment AB: $(AB) = \| \vec{u}_{1,0} - \vec{u}_{2,0} \|$. After trigonometric manipulations, the result is

$$\begin{aligned} \mathcal{D}(d_1, d_2) &= 2 * \sin(\theta/2) \\ &= \sqrt{2 * (1 - \cos(\theta))} \\ &= \sqrt{2 * (1 - similarity(d_1, d_2))} \end{aligned} \tag{11.5}$$

Notice that Eq. 11.5 defines a distance function (non-negative, symmetric, satisfying the triangular inequality) and that it decreases with increasing similarity.

Also notice that it allows us to respond to range queries: Suppose that the user wants all the documents d that are similar to the query document q:

$$similarity(d, q) \geq \delta$$

Thanks to Eq. 11.5, the requirement becomes

$$\mathcal{D}(d, q) \leq \sqrt{2 * (1 - \delta)} \tag{11.6}$$

which eventually becomes a range query in our 'target' space and can be handled efficiently by any SAM.

In [FL95], one of the testbeds we used was a collection of 35 text documents in 7 groups, with 5 documents per group: Abstracts of computer science technical

reports (labeled as 'Abs'), reports about basketball games ('Bbr'), 'call for papers' for technical conferences ('Cal'), portions of the Bible (from the Gospel of Matthew) ('Mat'), cooking recipes ('Rec'), 'world news' (documents about the Middle East - October 1994) ('Wor'), and sale advertisements for computers and software ('Sal')

Figure 11.7 shows the results of FastMap on the above dataset, with k=3 dimensions. The Figure shows the 3-d scatter-plot, (a) in its entirety and (b) after zooming into the center, to highlight the clustering abilities of FastMap. Notice that the 7 classes are separated well, in only k=3 dimensions. Additional experiments, involving real and synthetic datasets, are in [FL95].

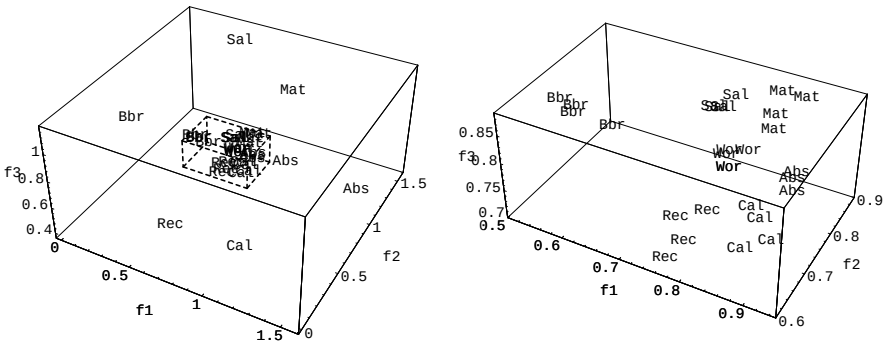


Figure 11.7 The DOCS dataset, after *FastMap* in k=3-d space (a) The whole collection (b) magnification of the dashed box.

11.5 CONCLUSIONS

FastMap is a linear algorithm that maps N objects into k-d points, with small distortion of the distances. Thus it provides an automated way to extract features, for a given dataset $\mathcal{O}$ and a given distance function $\mathcal{D}()$.

Mapping objects into points (while preserving the distances well) is vital for fast searching using the 'GEMINI' approach. Moreover, such a mapping is useful for data-mining, cluster analysis and visualization of a multimedia dataset. Notice

that FastMap is linear on the number of objects N, while Multidimensional Scaling (MDS) is quadratic, and thus impractical for large databases.

Exercises

Exercise 11.1 [30] *Implement MDS; apply it on some distance matrices*

Exercise 11.2 [30] *Implement the string-editing distance function; compute the distance matrix for 100 English words (eg., from* `/usr/dict/words`*), and map them into 2-d points using the MDS package of Exercise 11.1*

Exercise 11.3 [25] *Implement FastMap and compare it against MDS, with respect to its speed and its stress, on a set of 100, 200, 500 English words from* `/usr/dict/words`*. List your observations.*

12

CONCLUSIONS

We have presented a generic method (the '*GEMINI*' approach) to accelerate queries by content on image databases and, more general, on multimedia databases. Target queries are, eg., '*find images with a color distribution of a sunset photograph*'; or, '*find companies whose stock-price moves similarly to a given company's stock*'.

The method expects a distance function $\mathcal{D}()$ (given by domain experts), which should measure the dis-similarity between two images or objects O_A, O_B. We mainly focus on *whole match* queries (that is, *queries by example*, where the user specifies the ideal object and asks for all objects that are within distance ϵ from the ideal object). Extensions to other types of queries (nearest neighbors, 'all pairs' and sub-pattern match) are briefly discussed.

The '*GEMINI*' approach combines two ideas:

- The first is to devise a '*quick and dirty*' test, which will eliminate several non-qualifying objects. To achieve that, we should extract k numerical features from each object, which should somehow describe the object (for example, the first few DFT coefficients for a time sequence, or for a gray-scale image). The key question to ask is *'If we are allowed to use only one numerical feature to describe each data object, what should this feature be?'*
- The second idea is to further accelerate the search, by organizing these k-dimensional points using state of the art spatial access methods ('SAMs') like the R-trees. These methods typically group neighboring points together, thus managing to discard large un-promising portions of the address space early.

The above two ideas achieve fast searching. We went further, and we considered the condition under which the above method will be not only fast, but also *correct*, in the sense that it will not miss any qualifying object (false alarms are acceptable, because they can be discarded, with the obvious way). The answer is the *lower-bounding* lemma, which intuitively states that the mapping $\mathcal{F}()$ of objects to k-d points should *make things look closer*.

The rest of the chapters describe how to apply the method for a variety of environments, like 1-d time sequences and 2-d color images. These environments are specifically chosen, because they give rise to the 'dimensionality-curse' and the 'cross-talk' problems, respectively. The approach of the 'quick-and-dirty' filter, together with the lower-bounding lemma, provided solutions to both cases. Experimental results on real or realistic data illustrated the speed-up that the 'GEMINI' approach provides.

In the last two Chapters focused on two related problems. In Chapter 10 we presented a method to handle sub-pattern matching in time sequences, by using again a 'quick-and-dirty' filter and a sliding window, to map each sequence into a trail in feature space; the trail is represented coarsely by a small number of minimum bounding (hyper-)rectangles, which are fed into a Spatial Access Method (SAM). Finally, in Chapter 11 we discussed 'FastMap', a linear, approximate algorithm which can derive features automatically, given a set of objects $\mathcal{O}$ and a distance function $\mathcal{D}()$.

PART III

MATHEMATICAL TOOLBOX

A

PRELIMINARIES

Before we start, we need some definitions from complex algebra and from linear algebra. Consider a complex number

$$c = a + jb = A\exp(j\phi)$$

where $j = \sqrt{-1}$ is the imaginary unit. Then, we have the following:

Definition A.1 *The amplitude* $|c|$ *is defined as* $A \equiv |c| = \sqrt{a^2 + b^2}$

Definition A.2 *The phase* ϕ *of the number* $c = a + jb$ *is defined as*

$$\phi = \arctan(b/a)$$

Definition A.3 *The conjugate* c^* *of* c *is defined as* $a - jb$.

Definition A.4 *The energy* $E(c)$ *of* c *is defined as the square of the amplitude* $(E(c) \equiv |c|^2 \equiv c\ c^*)$.

From matrix algebra, we use the following notation and concepts. $\vec{x}$ will be considered as a column vector. Eg.,

$$\vec{x} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \tag{A.1}$$

In general, lower-case letters with an arrow will denote column vectors. Capital, bold letters (eg., $\mathbf{A}$) will denote matrices: Eg., $\mathbf{A} = [a_{i,j}]$ where i, j span the rows and columns, respectively.

Definition A.5 $\mathbf{A}^t$ *denotes the transpose of a matrix:* $\mathbf{A}^t = [a_{j,i}]$ *(notice the reversal of* i *and* j*). If the matrix* $\mathbf{A}$ *has complex entries, then we want the so-called hermitian matrix* $\mathbf{A}^* = [a^*_{j,i}]$.

Clearly, for real-valued matrices, the transpose matrix is the hermitian matrix.

Definition A.6 *The norm* $\| \vec{x} \|$ *of a vector* $\vec{x}$ *is its Euclidean norm (root of sum of squares).*

Definition A.7 *The energy* $E(\vec{x})$ *of a sequence (or vector)* $\vec{x}$ *is defined as the sum of energies at every point of the sequence:*

$$E(\vec{x}) \equiv \| \vec{x} \|^2 \equiv \sum_{i=0}^{n-1} |x_i|^2 \tag{A.2}$$

Obviously, the energy of a signal is the square of the Euclidean norm ($\equiv$ length) $\| \vec{x} \|$ of the vector $\vec{x}$.

Definition A.8 *The inner or 'dot' product* $\vec{x} \circ \vec{y}$ *of two vectors* $\vec{x}$ *and* $\vec{y}$ *is defined as*

$$\vec{x} \circ \vec{y} = \sum_i (x_i * y_i) = (\vec{x})^t \times \vec{y} \tag{A.3}$$

where '' denotes scalar multiplication and '*$\times$*' denotes matrix multiplication*

Obviously, $\| \vec{x} \|^2 = \vec{x} \circ \vec{x}$. For complex-valued vectors, we use the hermitian instead of the transpose.

Definition A.9 *Orthogonality: Two vectors* $\vec{x}$ *and* $\vec{y}$ *are orthogonal (* $\vec{x} - \vec{y}$ *) iff* $\vec{x} \circ \vec{y} = 0$

Definition A.10 *A matrix* $\mathbf{A} = [\vec{a_1}, \vec{a_2}, \ldots]$ *is column-orthonormal iff its column vectors* $\vec{a_i}$ *are unit vectors and mutually orthogonal, that is*

$$\vec{a_i} \circ \vec{a_j} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{otherwise} \end{cases} \tag{A.4}$$

The definition for row-orthonormal is symmetric. From the above definition, if $\mathbf{A}$ is a column-orthonormal matrix, then

$$\mathbf{A}^t \times \mathbf{A} = \mathbf{I} \tag{A.5}$$

where $\mathbf{I}$ is the identity matrix of the appropriate dimensions. Again, for complex-valued matrices, we use the hermitian instead of the transpose.

B

FOURIER ANALYSIS

B.1 DEFINITIONS

The intuition behind the Fourier Transform (as well as the Discrete Time Fourier transform that we are examining) is based on Fourier's theorem, that every continuous function can be considered as a sum of sinusoidal functions. For the discrete case, which is the one of interest to us, the n-point *Discrete Fourier Transform* [OS75] of a signal $\vec{x} = [x_i]$, $i = 0, \ldots, n-1$ is defined to be a sequence $\vec{X}$ of n complex numbers X_f, $f = 0, \ldots, n-1$, given by

$$X_f = 1/\sqrt{n} \sum_{i=0}^{n-1} x_i \exp(-j2\pi fi/n) \quad f = 0, 1, \ldots, n-1 \tag{B.1}$$

where j is the imaginary unit ($j \equiv \sqrt{-1}$).

Compactly,

$$\vec{x} \Longleftrightarrow \vec{X} \tag{B.2}$$

will denote a DFT pair. The signal $\vec{x}$ can be recovered by the inverse transform:

$$x_i = 1/\sqrt{n} \sum_{f=0}^{n-1} X_f \exp(j2\pi fi/n) \quad i = 0, 1, \ldots, n-1 \tag{B.3}$$

X_f is a complex number, with the exception of X_0, which is a real if the signal $\vec{x}$ is real. There are some minor discrepancies among books: some define $X_f = 1/n \sum_{i=0}^{n-1} \ldots$ or $X_f = \sum_{i=0}^{n-1} \ldots$. We have followed the definition in (Eq B.1), for it simplifies the upcoming Parseval's theorem (Eq B.5).

Recall that $\exp(j\phi) \equiv \cos(\phi) + j\sin(\phi)$. The intuition behind DFT is to decompose a signal into sine and cosine functions of several frequencies, multiples

of the basic frequency $1/n$. The reasons of its success is that certain operations, like filtering, noise removal, convolution and correlation, can be executed more conveniently in the frequency domain. Moreover, the frequency domain highlights some properties of the signals, such as periodicity.

It is instructive to envision the DFT as a matrix operation:

Observation B.1 *Eq. B.1 can be re-written as*

$$\vec{X} \;=\; \mathbf{A} \times \vec{x} \tag{B.4}$$

where $\mathbf{A} = [a_{i,f}]$ *is an* $n \times n$ *matrix with*

$$a_{i,f} = 1/\sqrt{n}\exp(-j2\pi fi/n) \quad i, f = 0, \ldots, n-1$$

Observation B.2 *Notice that* $\mathbf{A}$ *is* column-orthonormal, *that is, its column vectors are unit vectors, mutually orthogonal. It is also true that* $\mathbf{A}$ *is row-orthonormal, since it is a square matrix [PTVF92, p. 60].*

That is,

$$\mathbf{A}^* \times \mathbf{A} = \mathbf{A} \times \mathbf{A}^* = \mathbf{I}$$

where $\mathbf{I}$ is the $(n \times n)$ identity matrix and $\mathbf{A}^*$ is the conjugate-transpose ('*hermitian*') of $\mathbf{A}$, that is $\mathbf{A}^* = [a^*_{f,i}]$

The above observation has a very useful, geometric interpretation: since the DFT corresponds to a matrix multiplication with $\mathbf{A}$ of Eq. B.4, and since the matrix $\mathbf{A}$ is orthonormal, the matrix $\mathbf{A}$ effectively does a rotation (but *no scaling*) of the vector $\vec{x}$ in n-dimensional complex space; as a rotation, *it does not affect the length* of the original vector, nor the Euclidean distance between any pair of points.

B.2 PROPERTIES OF DFT

Next we list the properties of DFT that are most useful for our applications.

Theorem 1 (Parseval) *Let $\vec{X}$ be the Discrete Fourier Transform of the sequence $\vec{x}$. Then we have*

$$\sum_{i=0}^{n-1} |x_i|^2 = \sum_{f=0}^{n-1} |X_f|^2 \tag{B.5}$$

Proof: See, eg., [OS75]. □

An easier proof is based on Observation B.2 above. Intuitively, Parsevals' theorem states that the DFT preserves the energy (= square of the length) of the signal.

Property B.1 *The DFT also preserves the Euclidean distance.*

Proof: Using the fact that the DFT is equivalent to matrix multiplication (Eq. B.4), and that the matrix $\mathbf{A}$ is column-orthonormal, we can prove that the DFT also preserves the distance between two signals $\vec{x}$ and $\vec{y}$. Let $\vec{X}$ and $\vec{Y}$ denote their Fourier transforms. Then, we have:

$$\begin{aligned} \| \vec{X} - \vec{Y} \|^2 &= \| \mathbf{A} \times \vec{x} - \mathbf{A} \times \vec{y} \|^2 \\ &= (\mathbf{A} \times \vec{x} - \mathbf{A} \times \vec{y})^* \times (\mathbf{A} \times \vec{x} - \mathbf{A} \times \vec{y}) \\ &= (\vec{x} - \vec{y})^* \times \mathbf{A}^* \times \mathbf{A} \times (\vec{x} - \vec{y}) \\ &= (\vec{x} - \vec{y})^* \times \mathbf{I} \times (\vec{x} - \vec{y}) \\ &= \| \vec{x} - \vec{y} \|^2 \end{aligned} \tag{B.6}$$

That is, the DFT maintains the Euclidean distance between the two signals $\vec{x}$ and $\vec{y}$. □

Observation B.2 gives a strong intuitive 'proof' for the above two properties. The geometric point of view of Observation B.2 is important: *any* transformation that corresponds to an orthonormal matrix $\mathbf{A}$ will also enjoy a theorem similar to Parseval's theorem for the DFT. Such transformations are the DCT and the DWT, that we examine later.

Property B.2 *A shift in the time domain changes only the phase of the DFT coefficients, but not the amplitude:*

$$[x_{i-i_0}] \Longleftrightarrow [X_f \ \exp(-2\pi f i_0 j/n)] \tag{B.7}$$

where '$\Longleftrightarrow$' denotes a DFT pair. This is a useful property, if we are looking for, eg., matching stock prices with time-lags.

Property B.3 *For real signals, we have* $X_f = X^*_{n-f}$, *for* $f = 1, 2, \ldots, n-1$.

Proof: See [PTVF92, p. 511]. □

Using the above property, we only need to plot the amplitudes up to the middle, and specifically, up to q, if $n = 2q + 1$, or $q + 1$, if the duration is $n = 2q$.

Definition B.1 *The resulting plot of* $|X_f|$ *versus* f *will be called the amplitude spectrum or plain spectrum of the given time sequence; its square will be the energy spectrum or power spectrum.*

The phase spectrum is defined similarly, but it is not used as much as the amplitude and energy spectra.

Property B.4 *The DFT requires* $O(n \log n)$ *computation time.*

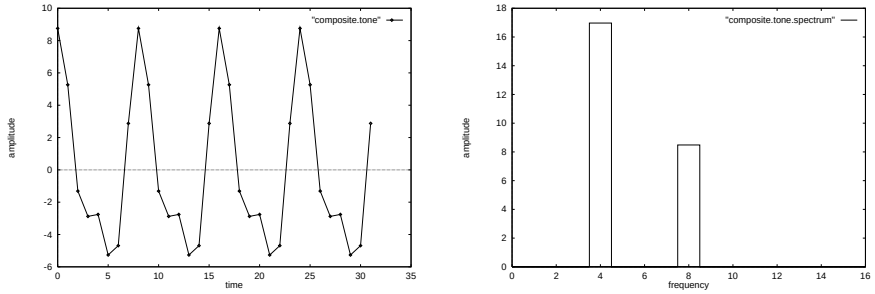
Although a straightforward computation of the DFT coefficients requires $O(n^2)$ time, the celebrated *Fast Fourier Transform* (FFT) exploits regularities of the $e^{j2\pi fi/n}$ function, and achieves $O(n \log n)$ time (eg., see [PTVF92]).

B.3 EXAMPLES

The main point we want to illustrate is the ability of the DFT to highlight the periodicities of the input signal $\vec{x}$. This is achieved through the *amplitude spectrum* that we defined before. Next, we give some carefully selected signals and their spectra.

Example B.1 *A composite tone:*

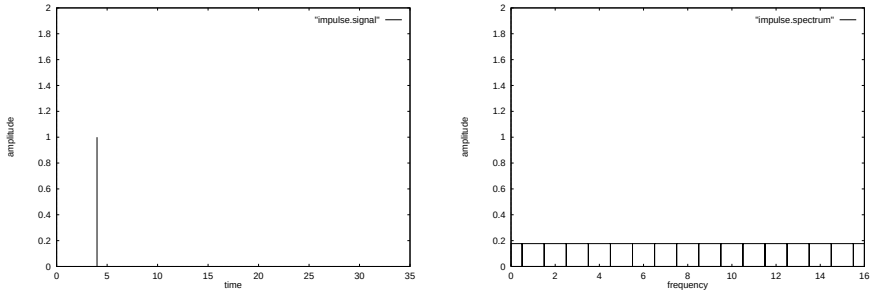
$$x_i = 6 \sin(2\pi 4i/n + 0.5) + 3 \sin(2\pi 8i/n) \quad i = 0, \ldots, 31 \tag{B.8}$$



(a) time domain (b) amplitude spectrum

Figure B.1 A composite tone and its amplitude spectrum. Notice the spikes for the two component frequencies at f=4 and 8.

This is a sum of sinusoidal functions of frequencies 4 and 8. Digitized voice and musical sounds are sums of a few sinusoidal functions. Figure B.1 shows the signal in the time domain, and its amplitude spectrum. In the latter, notice the two clear peaks, at frequencies 4 and 8, with amplitudes proportional to 6 and 3, respectively, with a constant of proportionality: $\sqrt{n}/2$.



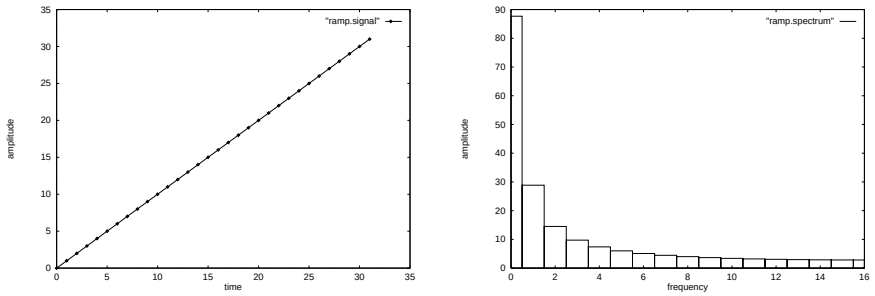
(a) time domain (b) amplitude spectrum

Figure B.2 A (shifted) impulse function, and its amplitude spectrum. Notice the 'frequency leak' in all the frequencies.

Example B.2 *The 'impulse function', or 'Dirac delta' function:* x_0 *=1;* x_i*=0 for* $i > 0$.

Figure B.2 shows an impulse function (shifted to the right, for better plotting), along with its spectrum. Notice that there is no dominating frequency in the amplitude spectrum. This is expected, since the original signal has no periodicities. This phenomenon is informally called *frequency leak*: the given signal has strong components on every frequency, and thus it can not be approximated (compressed) well by using few DFT coefficients.

In general, spikes and discontinuities require all the frequencies. Since the DFT effectively assumes that the signal is repeated infinite times, periodically (with period n), a high difference between x_0 and x_{n-1} also leads to frequency leak. This is illustrated with the next example:



(a) time domain (b) amplitude spectrum

Figure B.3 The ramp function, and its amplitude spectrum. Notice the frequency leak, again.

Example B.3 *The 'ramp' function:*

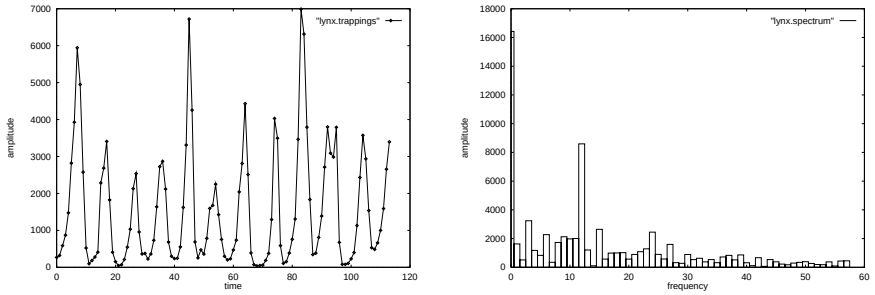
$$x_i = i$$

Figure B.3 shows the the ramp function and its DFT. Notice that it also has a 'frequency leak', having non-zero amplitudes for all its DFT coefficients. However, the amplitudes are decreasing, compared to the impulse function.

This is an important observation: in general, if the input signal has a trend, the DFT has a frequency leak. Notice that the upcoming DCT avoids this specific problem.

Example B.4 *Number of trappings for Canadian lynx (animals per year, 1821-1934).*

See Figure B.4. This is a well-known dataset in population ecology [Sig93, p. 45], as well as time-sequence analysis [Ton90, BJR94]. It has a strong periodical nature, which is highlighted by the spike in the spectrum. It is interesting to notice that the population of hares (not shown here) also follows a similar periodic pattern, with the same period, but with some phase lead, because hares are a major food source for the Canadian lynx.



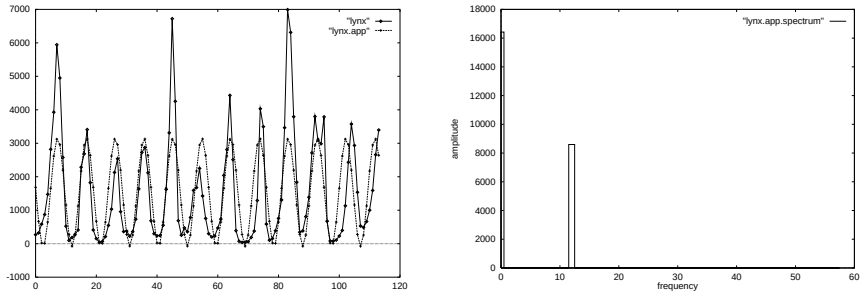
(a) time domain (b) amplitude spectrum

Figure B.4 Canadian lynx trappings (1821-1934), and its amplitude spectrum. Notice the spike at f=12, corresponding to a period of 9.5 years.

Figure B.5 highlights the ability of the DFT to concentrate the energy. Consider the spectrum of Figure B.4(b) and set to zero all the amplitudes, except for the two strongest ones (for f=0 and 12); this is the spectrum illustrated in Figure B.5(b). Figure B.5(a) shows the corresponding sequence in the time domain (by doing the inverse DFT), as well as the original 'lynx' dataset. Notice how well the approximate sequence matches the original, despite the tiny number of coefficients kept.

B.4 DISCRETE COSINE TRANSFORM (DCT)

For our purposes, the ideal transform should concentrate the energy into as few coefficients as possible, for most of the signals of interest. For several real signals, successive values are correlated: eg., in images, if a pixel is dark, chances are that its neighbors will also be dark. In these cases, the Discrete Cosine Transform (DCT) achieves better energy concentration than the DFT, and very close to optimal [Gal91, p. 54].



(a) (b)

Figure B.5 (a) The original lynx dataset (with 'diamonds') and its approximation (with 'crosses') (b) the spectrum of the approximation.

Moreover, the DCT avoids 'frequency leak' problems that plague the DFT when the input signal has a 'trend' (see Example B.3). The DCT solves this problem cleverly, by conceptually reflecting the original sequence in the time axis around the last point and taking the DFT on the resulting, twice-as-long sequence. Exactly because the (twice-as-long) signal is symmetric, all the coefficients will be *real* numbers. Moreover, from the property B.3 of the DFT, their amplitudes will be symmetric along the middle ($X_f = X_{2n-f}$). Thus, we need to keep only the first n of them.

The formulas for the DCT are

$$X_f = 1/\sqrt{n} \sum_{i=0}^{n-1} x_i \cos \frac{\pi f(i+0.5)}{n} \quad f = 0, \ldots, n-1 \tag{B.9}$$

and for the inverse:

$$x_i = 1/\sqrt{n} X_0 + 2/\sqrt{n} \sum_{f=1}^{n-1} X_f \cos \frac{\pi f(i+0.5)}{n} \quad i = 0, \ldots, n-1 \tag{B.10}$$

As with the DFT, the complexity of the DCT is also $O(n \log(n))$.

B.5 m-DIMENSIONAL DFT/DCT (JPEG)

All the above transforms can be extended to m-dimensional signals: for m=2 we have gray-scale images, for m=3 we have 3-d MRI brain scans etc. Informally,

we have to do the transformation along each dimension: for example, for the DFT for a 1024x1024 matrix (eg., image), we have to do the DFT on each row, and then do the DFT on each column. Formally, for an $n_1 \times n_2$ array $[x_{i_1,i_2}]$ these operations are expressed as follows:

$$X_{f_1,f_2} = \frac{1}{\sqrt{n_1}} \frac{1}{\sqrt{n_2}} \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} x_{i_1,i_2} \exp\left(-2\pi j i_1 f_1/n_1\right) \exp\left(-2\pi j i_2 f_2/n_2\right)$$

where x_{i_1,i_2} is the value (eg., gray scale) of the position (i_1, i_2) of the array, and f_1, f_2 are the spatial frequencies, ranging from 0 to (n_1-1) and (n_2-1) respectively.

The formulas for higher dimensionalities m are straightforward. The formulas for the m-d inverse DFT and DCT are analogous. Notice that the 2-dimensional DCT is used in the JPEG standard [Wal91, Jur92] for image and video compression.

B.6 CONCLUSIONS

We have discussed some powerful, classic tools from signal processing, namely the DFT and DCT. The DFT is helpful in highlighting periodicities in the input signal, through its amplitude spectrum. The DCT is closely related to the DFT, and it has some additional desirable properties: its coefficients are always real (as opposed to complex), it handles well signals with trends, and it is very close to the optimal for signals whose successive values are highly correlated.

C

WAVELETS

C.1 MOTIVATION

The wavelet transform is believed to avoid the 'frequency leak' problem even better. Consider the case of an impulse function (Example B.2): both in the DFT and the DCT transform, it has non-zero amplitudes in all frequencies. Thus, what would take a single number to describe in the time domain, will require several numbers in the frequency domain. The problem is that the DFT has no temporal locality: each of its coefficients provide information about all the time instants. A partial remedy would be the so-called 'Short Window Fourier Transform' (SWFT) [RV91]: We can divide the time sequence into frames of, say, w consecutive (non-overlapping) samples, and do the w-point DFT in each of these windows. Thus, an impulse function in the time domain will have a restricted 'frequency leak'. Figure C.1 shows intuitively what happens: In the time domain, each value gives the full information about that instant (but no information about frequencies). The DFT has coefficients that give full information about a given frequency, but it needs all the frequencies to recover the value at a given instant in time. The SWFT is somewhere in between.

The only non-elegant point of the SWFT is the choice of the width w of the window: How large should it be, and why? The solution to this problem is very clever: let the width w be variable! This is the basis for the Discrete Wavelet Transform (DWT). Figure C.1 illustrates how the DWT coefficients tile the frequence-time plane.

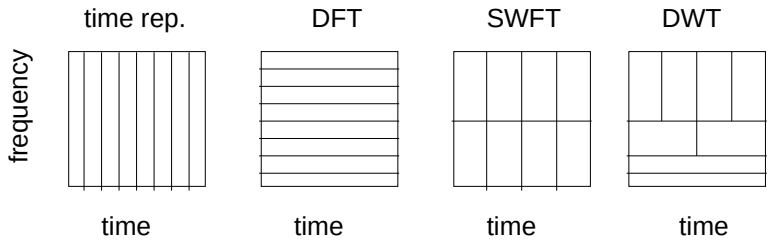


Figure C.1 Tilings of the time-frequency plane: (a) Time-domain representation (b) Discrete Fourier transform (DFT) (c) Short-Window Fourier transform (SWFT) (d) Discrete Wavelet transform (DWT).

C.2 DESCRIPTION

Several Discrete Wavelet transforms that have been proposed. The simplest to describe and code is the *Haar* transform. *Ignoring* temporarily some proportionality constants, the Haar transform operates on the whole signal, giving the sum and the difference of the left and right part; then it focuses recursively on each of the halves, and computes the difference of their two sub-halves, etc, until it reaches an interval with one only sample in it.

It is instructive to consider the equivalent, bottom-up procedure. The input signal $\vec{x}$ must have a length n that is a power of 2, by appropriate zero-padding if necessary.

1. Level 0: take the first two sample points x_0 and x_1, and compute their sum $s_{0,0}$ and difference $d_{0,0}$; do the same for all the other pairs of points (x_{2i}, x_{2i+1}). Thus, $s_{0,i} = C * (x_{2i} + x_{2i+1})$ and $d_{0,i} = C * (x_{2i} - x_{2i+1})$, where C is a proportionality constant, to be discussed soon. The values $s_{0,i}$ $(0 \leq i \leq n/2)$ constitute a 'smooth' (=low frequency) version of the signal, while the values $d_{0,i}$ represent the high-frequency content of it.
2. Level 1: consider the 'smooth' $s_{0,i}$ values; repeat the previous step for them, giving the even-smoother version of the signal $s_{1,i}$ and the smooth-differences $d_{1,i}$ $(0 \leq i \leq n/4)$
3. ... and so on recursively, until we have a smooth signal of length 2.

The Haar transform of the original signal $\vec{x}$ is the collection of all the 'difference' values $d_{l,i}$ at every level l and offset i, plus the smooth component $s_{L,0}$ at the last level L $(L = \log_2(n) - 1)$.

Following the literature, the appropriate value for the constant C is $1/\sqrt{2}$, because it makes the transformation matrix to be orthonormal (eg., see Eq. C.4). Adapting the notation (eg., from [Cra94] [VM]), the Haar transform is defined as follows:

$$d_{l,i} = 1/\sqrt{2}\ (s_{l-1,2i} - s_{l-1,2i+1}) \quad l = 0, \ldots, L, \quad i = 0, \ldots, n/2^{l+1} - 1 \quad \text{(C.1)}$$

with

$$s_{l,i} = 1/\sqrt{2}\ (s_{l-1,2i} + s_{l-1,2i+1}) \quad l = 0, \ldots, L, \quad i = 0, \ldots, n/2^{l+1} - 1 \quad \text{(C.2)}$$

with the initial condition:

$$s_{-1,i} = x_i \quad \text{(C.3)}$$

For example, the 4-point Haar transform is as follows. Envisioning the input signal $\vec{x}$ as a column vector, and its Haar transform $\vec{w}$ as another column vector ($\vec{w} = [s_{1,0}, d_{1,0}, d_{0,0}, d_{0,1}]^t$), the Haar transform is equivalent to a matrix multiplication, as follows:

$$\begin{bmatrix} s_{1,0} \\ d_{1,0} \\ d_{0,0} \\ d_{0,1} \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \text{(C.4)}$$

The above procedure is shared among *all* the wavelet transforms: We start at the lowest level, applying two functions at successive windows of the signal: the first function does some smoothing, like a weighted average, while the second function does a weighted differencing; the smooth (and shortened) version of the signal is recursively fed back into the loop, until the resulting signal is too short.

The Haar transform is still criticized for 'frequency leak' problems [Dau92, p. 10]. One of the most popular wavelet transforms is the so-called Daubechies-4 [Dau92]. We describe the derivation of the 0-th level of coefficients only, because the rest of the levels are derived recursively, as explained above. At each step, the Daubechies-4 DWT operates on 4 consecutive sample points; the 'smooth' component is given by

$$s_{0,i} = h_0 x_{2i} + h_1 x_{2i+1} + h_2 x_{2i+2} + h_3 x_{2i+3} \quad i = 0, \ldots, n/2 \quad \text{(C.5)}$$

and the 'difference' component is given by

$$d_{0,i} = h_3 x_{2i} - h_2 x_{2i+1} + h_1 x_{2i+2} - h_0 x_{2i+3} \quad i = 0, \ldots, n/2 \quad \text{(C.6)}$$

where

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}} \quad h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}} \quad h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}} \quad h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{C.7}$$

Notice that the signal is supposed to 'wrap-around' (ie., $x_{n+i} = x_i$ whenever the index i exceeds n). More details are in [PTVF92]. The code in `nawk` [AKW88] and Bourne 'shell' is attached in the next section.

Figure C.2 shows the basis functions of the Daubechies-4 DWT for n=32 points. The top left gives the basis function #5: it is the level-2 wavelet, starting at position 0. Notice that it mainly concentrates on the first half of the signal, giving a weighted difference. The top right gives the basis function #9: it is the level-1 wavelet starting at position 0. Notice that it has a shorter time-span than the previous (#5), but more violent oscillation (thus, higher frequency content). The bottom row shows the basis functions #17 and #18. They correspond to level-0 wavelets starting at offsets t=0 and t=2, respectively. As expected, the basis functions of this level have the shortest time-span and the highest frequency content. Also as expected, these two basis functions have identical shape and only differ by a horizontal shift.

C.3 DISCUSSION

The computational complexity of the above transforms is $O(n)$, as it can be verified from Eq. C.1-C.3. Notice that this is faster than the $O(n \log(n))$ of FFT, without even the need to resort to any of the FFT-like techniques.

In addition to their computational speed, there is a fascinating relationship between wavelets, multiresolution methods (like quadtrees or the pyramid structures in machine vision), and fractals. The reason is that wavelets, like quadtrees, will need only a few non-zero coefficients for regions of the image (or the time sequence) that are smooth/homogeneous, while they will spend more effort on the 'high activity' areas. It is believed [Fie93] that the mammalian retina consists of neurons which are tuned each to a different wavelet. Naturally occurring scenes tend to excite only few of the neurons, implying that a wavelet transform will achieve excellent compression for such images. Similarly, the human ear seems to use a wavelet transform to analyze a sound, at least in the very first stage [Dau92, p. 6] [WS93].

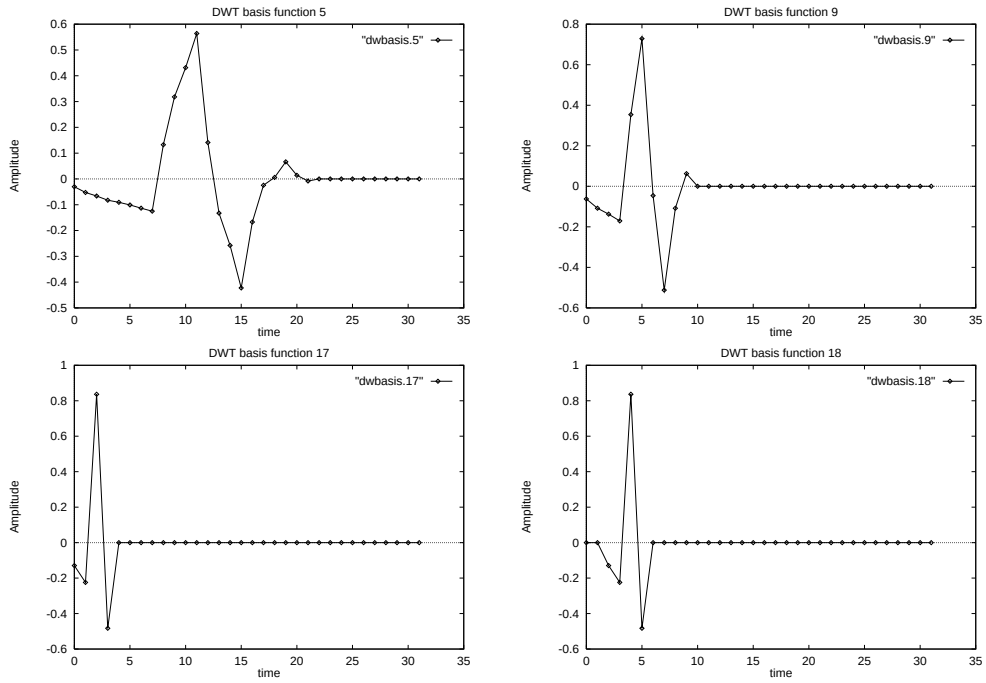


Figure C.2 Basis functions #5, 9, 17, 18, for the Daubechies-4 DWT.

C.4 CODE FOR DAUBECHIES-4 DWT

Next we give the source code for the Daubechies-4 DWT. We have chosen `nawk` [AKW88] because (a) it is a language on a higher level than 'C' and (b) it is more widely available than the even higher level languages, of mathematical packages (like 'Mathematica', 'Maple', 'MatLab' etc). See [PTVF92, VM] for wavelet code in some of the above languages. Object code for several wavelet transforms is available in the 'xwpl' package (`http: //www.math.yale.edu:80 /wavelets/`) from Yale.

```
#! /bin/sh -f

nawk '

# implements the Daubechies-4 wavelet -
# Following "Numerical Recipes in C", p. 593.
```

```
# Author: Christos Faloutsos, 1996
# Notice: the input signal MUST HAVE a length that is
#    a power of 2 (and greater or equal to 4)
# Expected input format: a sequence of numbers,
# separated by white space (tabs, blanks, newlines)

BEGIN{ c[0] = (1+sqrt(3))/(4*sqrt(2));
       c[1] = (3+sqrt(3))/(4*sqrt(2));
       c[2] = (3-sqrt(3))/(4*sqrt(2));
       c[3] = (1-sqrt(3))/(4*sqrt(2));
       TOL = 10^(-6);
       count = 0 ;
}
        function abs(xx) {
            res = xx
            if( xx < 0 ){ res =  -xx}
            return res
        }

        # chopArray
        function chopArray( xarg, Narg){
            for(ii=1; ii<=Narg; ii++){
                if( abs(xarg[ii]) <  TOL) {xarg[ii] =0}
            }
        }

        # print array
        function printArray ( x, N){
            for(i=1; i<=N; i++){ printf "%g\n", x[i] }
        }

        ######################################################
        # wraps the ivalue in the 1-Nval interval
        ######################################################
        function wrap ( ival, Nval) {
            resval = ival-1;
            resval = resval % Nval
            resval ++
            return resval
        }
        ######################################################
```

```
        ######################################################
        # performs one step of the DWT transform
        # on  array xarg[1:Narg]
        # Narg: should be a power of 2, and > 4
        # It does the changes IN PLACE
        ######################################################
        function oneStepDWT ( xarg, Narg ) {
            jj = 0;
            for( ii=1; ii<Narg; ii +=2 ){
                jj ++;
                sres[jj] = c[0]*xarg[wrap(ii,Narg)] + \
                           c[1]*xarg[wrap(ii+1,Narg)] + \
                           c[2]*xarg[wrap(ii+2,Narg)]+ \
                           c[3]*xarg[wrap(ii+3,Narg)];
                dres[jj] = c[3]*xarg[wrap(ii,Narg)] -  \
                           c[2]*xarg[wrap(ii+1,Narg)] + \
                           c[1]*xarg[wrap(ii+2,Narg)] -  \
                           c[0]*xarg[wrap(ii+3,Narg)];
            }
            for( ii=1; ii<= Narg/2; ii++ ){
                xarg[ii] = sres[ii];
                xarg[ii + Narg/2 ] = dres[ii]
            }
            return
        }
        ######################################################

        ######################################################
        # Does the full wavelet transform -
        # it calls repeatedly the oneStepDWT()
        # The array xarg[1,N] is changed IN PLACE
        ######################################################
        function DWT(xarg, Narg){
            # assert that Narg >= 4 and Narg: power of 2
            # WILL NOT WORK OTHERWISE
            for( len=Narg; len>=4; len = len/2){
                oneStepDWT(xarg, len)
            }
        }
        ######################################################

# read in the elements of the array
```

```
{ for( j = 1; j<= NF; j++) { count++; x[count] = $j } }

END {
    N =count; # array length
    DWT(x,N)
    chopArray(x,N)
    printArray(x,N)
}
' $*
```

C.5 CONCLUSIONS

The Discrete Wavelet Transform (DWT) achieves even better energy concentration than the DFT and DCT transforms, for natural signals [PTVF92, p. 604]. It uses multiresolution analysis, and it models well the early signal processing operations of the human eye and human ear.

D

K-L AND SVD

D.1 THE KARHUNEN-LOEVE (K-L) TRANSFORM

Before we examine the K-L transform, we should give the definition of eigenvalues and eigenvectors of a square matrix $\mathbf{S}$. (We use the letter 'S' to stress the fact that the matrix is square).

Definition D.1 *For a square $n \times n$ matrix $\mathbf{S}$, the unit vector $\vec{x}$ and the scalar λ that satisfy*

$$\mathbf{S} \times \vec{x} = \lambda \times \vec{x} \tag{D.1}$$

are called an eigenvector and its corresponding eigenvalue of the matrix $\mathbf{S}$.

The eigenvectors of a *symmetric* matrix are mutually orthogonal and its eigenvalues are real. See [PTVF92, p. 457] for more details.

The intuitive meaning of these concepts is the following: A matrix $\mathbf{S}$ defines an affine transformation $\vec{y} = \mathbf{S} \times \vec{x}$, that involves rotation and/or scaling; the eigenvectors are the unit vectors along the directions that are *not* rotated by $\mathbf{S}$; the corresponding eigenvalues show the scaling. For example, the matrix

$$\mathbf{S} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \tag{D.2}$$

gives the ellipse of Figure D.1, when applied to the periphery of the unit circle. The major and minor axes of the ellipse correspond to the directions of the

two eigenvectors of $\mathbf{S}$. Specifically, the strongest eigenvalue corresponds to the eigenvector of the major axis:

$$\lambda_1 = 3.62 \quad \vec{u_1} = \begin{bmatrix} 0.52 \\ 0.85 \end{bmatrix}; \qquad \lambda_2 = 1.38 \quad \vec{u_2} = \begin{bmatrix} 0.85 \\ -0.52 \end{bmatrix} \tag{D.3}$$

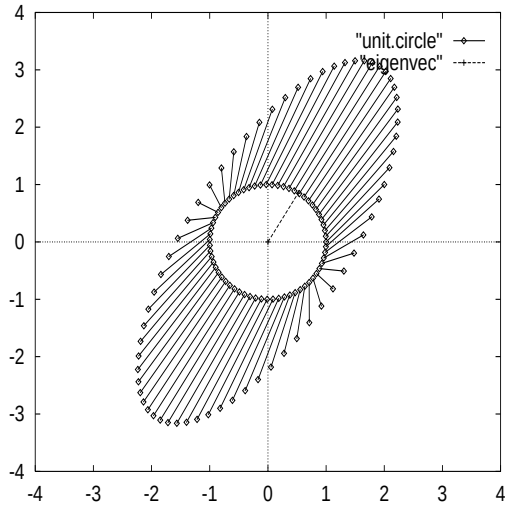


Figure D.1 The matrix $\mathbf{S}$ as a transformation: the unit circle becomes an ellipse, with major axis along the first eigenvector

The following observation will be used later, to show the strong connection between the eigenvalue analysis and the upcoming Singular Value Decomposition (Theorem 2).

Observation D.1 *If* $\mathbf{S}$ *is a real and symmetric matrix (ie.,* $\mathbf{S} = \mathbf{S}^t$*), then it can be written in the form*

$$\mathbf{S} = \mathbf{U} \times \mathbf{\Lambda} \times \mathbf{U}^t \tag{D.4}$$

where the columns of $\mathbf{U}$ *are the eigenvectors of* $\mathbf{S}$ *and* $\mathbf{\Lambda}$ *is a diagonal matrix, with values the corresponding eigenvalues of* $\mathbf{S}$*.*

Notice that the above observation is a direct consequence of the definition of the eigenvalues and eigenvectors (Definition D.1) and the fact that the eigenvectors are mutually orthogonal, or, equivalently, that $\mathbf{U}$ is column-orthonormal.

As an arithmetic example, for the matrix **S** that we used in the example (see Eq. D.2), we have

$$\mathbf{S} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 0.52 & 0.85 \\ 0.85 & -0.52 \end{bmatrix} \times \begin{bmatrix} 3.62 & 0 \\ 0 & 1.38 \end{bmatrix} \times \begin{bmatrix} 0.52 & 0.85 \\ 0.85 & -0.52 \end{bmatrix}$$

D.1.1 K-L: Problem definition

Consider the following problem: Given a collection of n-d points, project them on a k-d sub-space ($k \ll n$), minimizing the error of the projections (sum of squared differences). The problem has been studied extensively in statistical pattern recognition and matrix algebra. The optimal way to project n-dimensional points onto k-dimensional points ($k \leq n$) is the *Karhunen-Loève* ('K-L') transform (eg., see [DH73b], [Fuk90]). In other words, the K-L transforms gives linear combination of axis (='attributes' = 'features'), sorted in 'goodness' order.

Figure D.2 gives an illustration of the problem and the solution: it shows a set of 2-d points, and the corresponding 2 directions (x' and y') that the K-L transform suggests: If we are allowed only k=1, the best direction to project on is the direction of x'; the next best is y' etc.

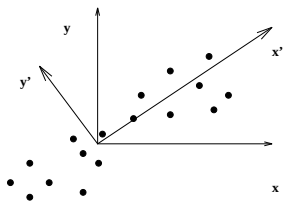


Figure D.2 Illustration of the Karhunen-Loève transformation - the 'best' axis to project is x'.

Next we give the detailed steps of the K-L, as well as the code in 'mathematica'. The K-L computes the eigenvectors of the *covariance* matrix (see Eq. D.7), sorts them in decreasing eigenvalue order, and approximates each data vector with its projections on the first k eigenvectors. The $n \times n$ covariance matrix **C** is defined as follows. Consider the $N \times n$ data matrix **A**, where rows correspond to data vectors and columns correspond to attributes. That is, $\mathbf{A} = [a_{ij}]$ ($i = 1, ...N$ and $j = 1, ...n$). The covariance matrix $\mathbf{C} = [c_{pq}]$ roughly gives the attribute-to-attribute similarity, by computing the un-normalized correlation between the

two attributes p and q. Let $\overline{a_{.,p}}$ be the average of the p-th column/attribute:

$$\overline{a_{.,p}} = 1/N \sum_{i=1}^{N} a_{ip} \quad p = 1, 2, ...n \tag{D.5}$$

Then, the entry $c_{p,q}$ of the covariance matrix $\mathbf{C}$ is

$$c_{p,q} = \sum_{i=1}^{N} (a_{i,p} - \overline{a_{.,p}})(a_{i,q} - \overline{a_{.,q}}) \tag{D.6}$$

Intuitively, we move the origin to the center of gravity of the given vectors, obtaining the matrix $\mathbf{B} = [b_{ij}] = [a_{ij} - \overline{a_{.,j}}]$. We shall refer to the matrix $\mathbf{B}$ as the 'zero-mean' matrix, exactly because its column averages are zero, by construction. Then, we compute the covariance matrix $\mathbf{C}$ as follows:

$$\mathbf{C} = \mathbf{B}^t \times \mathbf{B} \tag{D.7}$$

Example D.1 *Consider the data vectors* $\vec{a_1} = [1, 2]^t$, $\vec{a_2} = [1, 1]^t$ *and* $\vec{a_3} = [0, 0]^t$. *Then we have for the data matrix* $\mathbf{A}$*:*

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}$$

The column averages are the coordinates of the center of gravity:

$$\overline{a_{.,1}} = 2/3 \quad \overline{a_{.,2}} = 1$$

The 'zero-mean' matrix $\mathbf{B}$ *is*

$$\mathbf{B} = \begin{bmatrix} 1/3 & 1 \\ 1/3 & 0 \\ -2/3 & -1 \end{bmatrix}$$

and the covariance matrix $\mathbf{C}$ *is*

$$\mathbf{C} = \begin{bmatrix} 2/3 & 1 \\ 1 & 2 \end{bmatrix}$$

with eigenvalues and eigenvectors:

$$\lambda_1 = 2.53 \quad (\vec{u_1})^t = [0.47, 0.88]$$

$$\lambda_2 = 0.13 \quad (\vec{u_2})^t = [-0.88, 0.47]$$

Figure D.3 plots the 3 given points in 2-d space as 'diamonds', as well as their center of gravity (2/3, 1) and the corresponding K-L directions. It uses 'crosses' and 'squares' for the major and minor eigenvector, respectively.

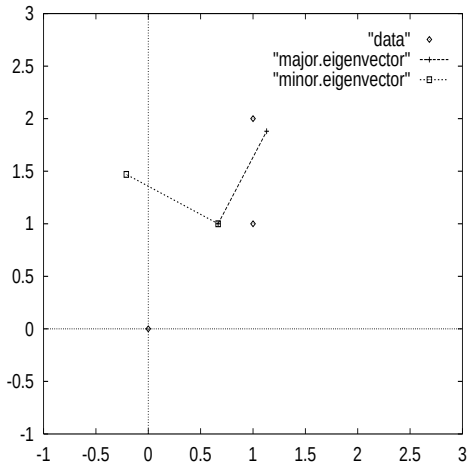


Figure D.3 Example of the Karhunen-Loève transformation, with the 3 points of the Example.

Next we give an implementation of the K-L transform in *Mathematica* [Wol91]. The major step is to compute the covariance matrix **C**; then, the eigenvalue routine `Eigensystem` does the rest (and hardest!) of the job.

```
(* given a matrix mat_ with $n$ vectors (rows) of $m$ attributes (columns),
   it creates a matrix with $n$ vectors and their
   first $k$ most 'important' attributes
   (ie., the K-L expansions of these $n$ vectors) *)
KLexpansion[ mat_, k_:2] := mat . Transpose[ KL[mat, k] ];

(* given a matrix with $n$ vectors of $m$ dimensions,
   computes the first $k$ singular vectors,
   ie., the axes of the first $k$ Karhunen-Lo\`{e}ve expansion *)
KL[ mat_ , k_:2 ]:= Module[
    {n,m, avgvec, newmat,i, val, vec },

    {n,m} = Dimensions[mat];
    avgvec = Apply[ Plus, mat] / n //N;
```

```
    (* translate vectors, so the mean is zero *)
    newmat = Table[ mat[[i]] - avgvec , {i,1,n} ];

    {val, vec} = Eigensystem[ Transpose[newmat] . newmat ];
    vec[[ Range[1,k] ]]
]
```

D.2 SVD

As we saw, the eigenvalues and eigenvectors are defined for square matrices. For rectangular matrices, a closely related concept is the *Singular Value Decomposition* (SVD) [Str80, PFTV88, GVL89]: Consider a set of points as before, represented as a $N \times n$ matrix $\mathbf{A}$, as in Table D.2.

term / document	data	information	retrieval	brain	lung
CS-TR1	1	1	1	0	0
CS-TR2	2	2	2	0	0
CS-TR3	1	1	1	0	0
CS-TR4	5	5	5	0	0
MED-TR1	0	0	0	2	2
MED-TR2	0	0	0	3	3
MED-TR3	0	0	0	1	1

Table D.1 Example of a (document-term) matrix

Such a matrix could represent, eg., N patients with n numerical symptoms each (blood pressure, cholesterol level etc), or N sales with n products in a data mining application [AS94], with the dollar amount spent on each product, by the given sale, etc. For concreteness, we shall assume that it represents N documents (rows) with n terms (columns) each, as happens in Information Retrieval (IR) [SFW83], [Dum94]. It would be desirable to group similar documents together, as well as similar terms together. This is exactly what SVD does, automatically! The only 'catch' is that SVD creates a *linear* combination of terms, as opposed to non-linear ones that, eg., Kohonen's neural networks could provide [LSM91, RMS92]. Nevertheless, these groups of terms are valuable: in Information Retrieval terminology, each would correspond to

a 'concept'; in the Karhunen-Loeve terminology, each group of terms would correspond to an important 'axes'. The formal definition for SVD follows:

Theorem 2 (SVD) *Given an $N \times n$ real matrix $\mathbf{A}$ we can express it as*

$$\mathbf{A} = \mathbf{U} \times \mathbf{\Lambda} \times \mathbf{V}^t \tag{D.8}$$

where $\mathbf{U}$ is a column-orthonormal $N \times r$ matrix, r is the rank of the matrix $\mathbf{A}$, $\mathbf{\Lambda}$ is a diagonal $r \times r$ matrix and $\mathbf{V}$ is a column-orthonormal $k \times r$ matrix.

Proof: See [PTVF92, p. 59]. □

The entries of $\mathbf{\Lambda}$ are non-negative. If we insist that the diagonal matrix $\mathbf{\Lambda}$ has its elements sorted in descending order, then the decomposition is *unique*[1].

Recall that a matrix $\mathbf{U}$ is column-orthonormal iff its column vectors are mutually orthogonal and of unit length. Equivalently: $\mathbf{U}^t \times \mathbf{U} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Schematically, see Figure D.4.

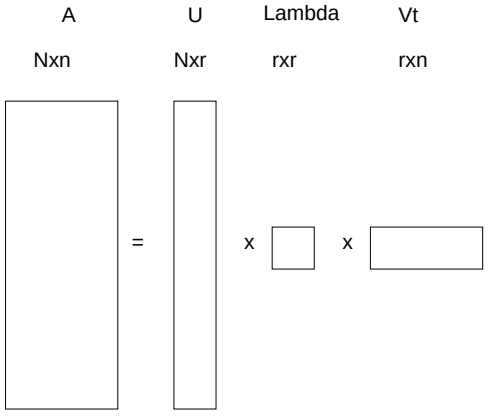


Figure D.4 Illustration of SVD

Eq. D.8 equivalently states that a matrix $\mathbf{A}$ can be brought in the form

$$\mathbf{A} = \lambda_1 \vec{u_1} \times (\vec{v_1})^t + \lambda_2 \vec{u_2} \times (\vec{v_2})^t + \ldots + \lambda_r \vec{u_r} \times (\vec{v_r})^t \tag{D.9}$$

[1] Except when there are equal entries in $\mathbf{\Lambda}$, in which case they and their corresponding columns of $\mathbf{U}$ and $\mathbf{V}$ can be permuted.

where $\vec{u_i}$, and $\vec{v_i}$ are column vectors of the $\mathbf{U}$ and $\mathbf{V}$ matrices respectively, and λ_i the diagonal elements of the matrix $\mathbf{\Lambda}$. Intuitively, the SVD identifies 'rectangular blobs' of related values in the $\mathbf{A}$ matrix. For example, for the above 'toy' matrix of Table D.2, we have two 'blobs' of values, while the rest of the entries are zero. This is confirmed by the SVD, which identifies them both:

$$\mathbf{A} = 9.64 \times \begin{bmatrix} 0.18 \\ 0.36 \\ 0.18 \\ 0.90 \\ 0 \\ 0 \\ 0 \end{bmatrix} \times [0.58,\ 0.58,\ 0.58,\ 0,\ 0] + 5.29 \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.53 \\ 0.80 \\ 0.27 \end{bmatrix} \times [0,\ 0,\ 0,\ 0.71,\ 0.71]$$

or

$$\mathbf{A} = \begin{bmatrix} 0.18 & 0 \\ 0.36 & 0 \\ 0.18 & 0 \\ 0.90 & 0 \\ 0. & 0.53 \\ 0. & 0.80 \\ 0. & 0.27 \end{bmatrix} \times \begin{bmatrix} 9.64 & 0 \\ 0 & 5.29 \end{bmatrix} \times \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix}$$

Notice that the rank of the matrix is r=2: there are effectively 2 types of documents (CS and Medical documents) and 2 'concepts', ie., groups-of-terms: the 'CS concept' (that is, the group {'data', 'information', 'retrieval'}), and the 'medical concept' (that is, the group {'lung', 'brain'}). The intuitive meaning of the $\mathbf{U}$ and $\mathbf{V}$ matrices is as follows: $\mathbf{U}$ can be thought of as the *document-to-concept* similarity matrix, while $\mathbf{V}$, symmetrically, is the *term-to-concept* similarity matrix. For example, $v_{1,2} = 0$ means that the first term ('data') has zero similarity with the 2nd concept (the 'lung-brain' concept).

The SVD is a powerful operation, with several applications. We list some observations, which are useful for multimedia indexing and Information Retrieval:

Observation D.2 *The $N \times N$ matrix $\mathbf{D} = \mathbf{A} \times \mathbf{A}^t$ will intuitively give the document-to-document similarities - in our case, it is*

$$\mathbf{D} = \mathbf{A} \times \mathbf{A}^t = \begin{bmatrix} 3 & 6 & 3 & 15 & 0 & 0 & 0 \\ 6 & 12 & 6 & 30 & 0 & 0 & 0 \\ 3 & 6 & 3 & 15 & 0 & 0 & 0 \\ 15 & 30 & 15 & 75 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 12 & 4 \\ 0 & 0 & 0 & 0 & 12 & 18 & 6 \\ 0 & 0 & 0 & 0 & 4 & 6 & 2 \end{bmatrix}$$

Observation D.3 *The eigenvectors of the* $\mathbf{D}$ *matrix will be the columns of the* $\mathbf{U}$ *matrix of the SVD of* $\mathbf{A}$.

Observation D.4 *Symmetrically, the $n \times n$ matrix $\mathbf{T} = \mathbf{A}^t \times \mathbf{A}$ will give the term-to-term similarities - in our example, it is:*

$$\mathbf{T} = \mathbf{A}^t \times \mathbf{A} \begin{bmatrix} 31 & 31 & 31 & 0 & 0 \\ 31 & 31 & 31 & 0 & 0 \\ 31 & 31 & 31 & 0 & 0 \\ 0 & 0 & 0 & 14 & 14 \\ 0 & 0 & 0 & 14 & 14 \end{bmatrix}$$

Observation D.5 *Similarly, the eigenvectors of the* $\mathbf{T}$ *matrix are the columns of the* $\mathbf{V}$ *matrix of the SVD of* $\mathbf{A}$.

Observation D.6 *Both* $\mathbf{D}$ *and* $\mathbf{T}$ *have the same eigenvalues, who are the squares of the λ_i elements of the* $\mathbf{\Lambda}$ *matrix of the SVD of* $\mathbf{A}$.

All the above observations can be proved from Theorem 2 and from the fact that $\mathbf{U}$ and $\mathbf{V}$ are column-orthonormal:

$$\mathbf{A} \times \mathbf{A}^t = \mathbf{U} \times \mathbf{\Lambda} \times \mathbf{V}^t \times \mathbf{V} \times \mathbf{\Lambda} \times \mathbf{U}^t = \mathbf{U} \times \mathbf{\Lambda} \times \mathbf{I} \times \mathbf{\Lambda} \times \mathbf{U}^t = \mathbf{U} \times \mathbf{\Lambda}^2 \times \mathbf{U}^t \quad \text{(D.10)}$$

and similarly for $\mathbf{A}^t \times \mathbf{A}$. According to (Eq. D.4), the columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{A} \times \mathbf{A}^t$, and its eigenvalues are the diagonal elements of $\mathbf{\Lambda}^2$ (that is, $\lambda_1^2, \ldots \lambda_r^2$).

The above observations illustrate the close relationship between the eigenvectors analysis of a matrix, the SVD and the K-L transform, which uses the eigenvectors of the covariance matrix $\mathbf{C}$ (Eq. D.7).

It should be noted that the SVD is extremely useful for several settings that involve least-squares optimization, such as in regression, in under-constraint and over-constraint linear problems, etc. See [PTVF92] or [Str80] for more details. Next, we show how it has been applied for Information Retrieval and filtering, under the name of *Latent Semantic Indexing* (LSI).

D.3 SVD AND LSI

Here we discuss SVD in more detail, in the context of Information Filtering. There, SVD has lead to the method of 'Latent Semantic Indexing' (LSI) [FD92b]. The idea is to try to group similar terms together, to form a few ($\approx 100 - 300$) 'concepts', and then map the documents into vectors in 'concept'-space, as opposed to vectors in n-dimensional space, where n is the vocabulary size of the document collection. This approach is a clever, automated way, to take into account term co-occurrences, building effectively a 'thesaurus without semantics': terms that often occur together, are grouped into 'concepts'; every time the user asks for a term, the system determines the relevant 'concepts' and searches for them.

In order to map document or query vectors into to concept space, we need the term-to-concept similarity matrix $\mathbf{V}$. For example, in the setting of Table D.2, consider the query *'find documents containing the term 'data'*". In this setting, this query $\vec{q}$ is the vector

$$\vec{q} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{D.11}$$

because the first entry corresponds to the term 'data' and the rest to the 4 other terms of our document collection (namely, 'information', 'retrieval', 'brain', 'lung'). To translate $\vec{q}$ to a vector $\vec{q_c}$ in concept space, we need the term-to-concept similarity matrix $\mathbf{V}$. The 'translation' is done if we multiply the query vector by $\mathbf{V}^t$:

$$\vec{q_c} \quad = \quad \mathbf{V}^t \times \vec{q}$$

$$= \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.58 \\ 0 \end{bmatrix}$$

which correctly corresponds to the fact that the query $\vec{q}$ is rather closely related to the CS group of terms (with 'strength' = 0.58), and unrelated to the medical group of terms ('strength' = 0). What is even more important is that the query $\vec{q_c}$ implicitly involves the terms 'information' and 'retrieval'; thus, an LSI-based system may return documents that do not necessarily contain the requested term 'data', but they are deemed relevant anyway. Eg., according to the running example, the document with the single word 'retrieval' will have the document vector $\vec{d}$

$$\vec{d} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{D.12}$$

which will be mapped to the concept vector $\vec{d_c}$:

$$\vec{d_c} = \mathbf{V}^t \times \vec{d_c} = \begin{bmatrix} 0.58 \\ 0 \end{bmatrix} = \vec{q_c} \tag{D.13}$$

That is, the above document will be a perfect match for the query ('data'), although it does not contain the query word.

Thanks to its ability to create a thesaurus of co-occurring terms, the LSI method has shown good performance. Experiments in [FD92b] report that LSI has equaled or outperformed standard vector methods and other variants, with improvement of as much as 30% in terms of precision and recall.

D.4 CONCLUSIONS

We have seen some powerful tools, based on eigenvalue analysis. Specifically:

- the 'K-L' transform is the optimal way to do dimensionality reduction: given N vectors with n dimensions, it provides the k most important directions on which to project (k is user defined).
- the SVD (singular value decomposition) operates on an $N \times n$ matrix and groups its rows and columns into r 'similar' groups, sorted in 'strength' order.

Both tools are closely related to the eigenvalue analysis (Eq. D.1): the K-L transform uses the eigenvalues of the covariance matrix; the SVD of a symmetric matrix is identical to its eigenvalue decomposition.

REFERENCES

[AC75] A.V. Aho and M.J. Corasick. Fast pattern matching: an aid to bibliographic search. *CACM*, 18(6):333–340, June 1975.

[ACF+93] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. Qbism: a prototype 3-d medical image database system. *IEEE Data Engineering Bulletin*, 16(1):38–42, March 1993.

[ACF+94] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. Qbism: Extending a dbms to support 3d medical images. *Tenth Int. Conf. on Data Engineering (ICDE)*, pages 314–325, February 1994.

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms (FODO) Conference*, Evanston, Illinois, October 1993. also available through anonymous ftp, from olympos.cs.umd.edu: ftp/pub/TechReports/fodo.ps.

[AGI+92] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Bala Iyer, and Arun Swami. An interval classifier for database mining applications. *VLDB Conf. Proc.*, pages 560–573, August 1992.

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[AIS93a] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: a performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[AIS93b] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD*, pages 207–216, May 1993.

[AKW88] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison Wesley, 1988.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. *Proc. of VLDB Conf.*, pages 487–499, September 1994.

[BB82] D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.

[BCC94] Eric W. Brown, James P. Callan, and W. Bruce Croft. Fast incremental indexing for full-text information retrieval. *Proc. of VLDB Conf.*, pages 192–202, September 1994.

[Ben75] J.L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *CACM*, 18(9):509–517, September 1975.

[Ben79] Jon L. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. on Software Engineering (TSE)*, SE-5(4):333–340, July 1979.

[BF95] Alberto Belussi and Christos Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. *Proc. of VLDB*, pages 299–310, September 1995.

[Bia69] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Trans. on Information Theory*, IT-15(6):658–664, November 1969.

[BJR94] George E.P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, NJ, 1994. 3rd Edition.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. *Proc. of ACM SIGMOD*, pages 237–246, May 1993.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322–331, May 1990.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. *ACM SIGMOD*, pages 197–208, May 1994.

[BM72] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[BM77] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *CACM*, 20(10):762–772, October 1977.

[But71] Arthur R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Trans. on Computers*, C-20(4):424–426, April 1971.

[BYG92] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Comm. of ACM (CACM)*, 35(10):74–82, October 1992.

[CE92] M. Castagli and S. Eubank. *Nonlinear Modeling and Forecasting.* Addison Wesley, 1992. Proc. Vol. XII.

[CF84] S. Christodoulakis and C. Faloutsos. Design considerations for a message file server. *IEEE Trans. on Software Engineering*, SE-10(2):201–210, March 1984.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Comm. of ACM*, 13(6):377–387, 1970.

[Coo70] W.S. Cooper. On deriving design equations for information retrieval systems. *JASIS*, pages 385–395, November 1970.

[CoPES92] Mathematical Committee on Physical and NSF Engineering Sciences. *Grand Challenges: High Performance Computing and Communications.* National Science Foundation, 1992. The FY 1992 U.S. Research and Development Program.

[CP90] Doug Cutting and Jan Pedersen. Optimizations for dynamic inverted index maintenance. *Proc. SIGIR*, pages 405–411, 1990.

[Cra94] Richard E. Crandall. *Projects in Scientific Computation.* Springer-Verlag New York, Inc., 1994.

[CS89] W.W. Chang and H.J. Schek. A signature access method for the starbust database system. In *Proc. VLDB Conference*, pages 145–153, Amsterdam, Netherlands, August 1989.

[CSY87] Shi-Kuo Chang, Qing-Yun Shi, and Cheng-Wen Yan. Iconic Indexing by 2-D Strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3):413–428, May 1987.

[CTH+86] S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria. Multimedia document presentation, information extraction and document formation in minos: a model and a system. *ACM TOOIS*, 4(4), October 1986.

[Dat86] Chris J. Date. *An Introduction to Database Systems.* Addison-Wesley, 1986. Vol. I; 4th ed.

[Dau92] Ingrid Daubechies. *Ten Lectures on Wavelets*. Capital City Press, Montpelier, Vermont, 1992. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[DDF+90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, September 1990.

[DH73a] R. Duda and P Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[DH73b] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[Dum94] Susan T. Dumais. Latent semantic indexing (lsi) and trec-2. In D. K. Harman, editor, *The Second Text Retrieval Conference (TREC-2)*, pages 105–115, Gaithersburg, MD, March 1994. NIST. Special publication 500-215.

[Dye82] C.R. Dyer. The space efficiency of quadtrees. *Computer Graphics and Image Processing*, 19(4):335–348, August 1982.

[Eli75] P. Elias. Universal codeword sets and representations of integers. *IEEE Trans. on Information Theory*, IT-21:194–203, 1975.

[EM66] Robert D. Edwards and John Magee. *Technical Analysis of Stock Trends*. John Magee, Springfield, Massachusetts, 1966. 5th Edition, second printing.

[EMS+86] J. Ewing, S. Mehrabanzad, S. Sheck, D. Ostroff, and B. Shneiderman. An experimental comparison of a mouse and arrow-jump keys for an interactive encyclopedia. *Int. Journal of Man-Machine Studies*, 24(1):29–45, January 1986.

[Equ93] W. Equitz. Retrieving images from a database using texture — algorithms from the QBIC system. Research report, IBM Almaden Research Center, San Jose, CA, 1993.

[Fal85] C. Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, March 1985.

[Fal88] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Trans. on Software Engineering*, 14(10):1381–1393, October 1988. early version available as UMIACS-TR-87-4, also CS-TR-1796.

[Fal92a] C. Faloutsos. Analytical results on the quadtree decomposition of arbitrary rectangles. *Pattern Recognition Letters*, 13(1):31–40, January 1992.

[Fal92b] Christos Faloutsos. Signature files. In William Bruce Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intell. Inf. Systems*, 3(3/4):231–262, July 1994.

[FBY92] W. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[FC87] C. Faloutsos and S. Christodoulakis. Optimal signature extraction and information loss. *ACM TODS*, 12(3):395–428, September 1987.

[FD92a] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35(12):51–60, December 1992.

[FD92b] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: an analysis of information filtering methods. *Comm. of ACM (CACM)*, 35(12):51–60, December 1992.

[FG96] Christos Faloutsos and Volker Gaede. Analysis of the z-ordering method using the hausdorff fractal dimension. *VLDB*, September 1996.

[FH69] J.R. Files and H.D. Huskey. An information retrieval system based on superimposed coding. *Proc. AFIPS FJCC*, 35:423–432, 1969.

[Fie93] D.J. Field. Scale-invariance and self-similar 'wavelet' transforms: an analysis fo natural scenes and mammalian visual systems. In M. Farge, J.C.R. Hunt, and J.C. Vassilicos, editors, *Wavelets, Fractals, and Fourier Transforms*, pages 151–193. Clarendon Press, Oxford, 1993.

[FJM94] Christos Faloutsos, H.V. Jagadish, and Yannis Manolopoulos. Analysis of the n-dimensional quadtree decomposition for arbitrary hyper-rectangles. CS-TR-3381, UMIACS-TR-94-130, Dept. of Computer Science, Univ. of Maryland, College Park, MD, December 1994. to appear in IEEE TKDE.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond Uniformity and Irndependence: Analysis of R-trees Using the Concept of Fractal Dimension. *Proc. ACM SIGACT-SIGMOD-SIGART PODS*, pages 4–13, May 1994. Also available as CS-TR-3198, UMIACS-TR-93-130.

[FL95] Christos Faloutsos and King-Ip (David) Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *Proc. of ACM-SIGMOD*, pages 163–174, May 1995.

[FN75] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. on Computers (TOC)*, C-24(7):750–753, July 1975.

[FNPS79] R. Fagin, J. Nievergelt, N. Pippenger, and H.R. Strong. Extendible hashing - a fast access method for dynamic files. *ACM TODS*, 4(3):315–344, September 1979.

[FR89a] C. Faloutsos and W. Rego. Tri-cell: a data structure for spatial objects. *Information Systems*, 14(2):131–139, 1989. early version available as UMIACS-TR-87-15, CS-TR-1829.

[FR89b] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.

[FR91] C. Faloutsos and Y. Rong. Dot: a spatial access method using fractals. In *IEEE Data Engineering Conference*, pages 152–159, Kobe, Japan, April 1991. early version available as UMIACS-TR-89-31, CS-TR-2214.

[Fre60] E. Fredkin. Trie memory. *CACM*, 3(9):490–500, September 1960.

[FRM94] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *Proc. ACM SIGMOD*, pages 419–429, May 1994. 'Best Paper' award; also available as CS-TR-3190, UMIACS-TR-93-131, ISR TR-93-86.

[FSN+95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jon Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: the qbic system. *IEEE Computer*, 28(9):23–32, September 1995.

[FSR87] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of object oriented spatial access methods. *Proc. ACM SIGMOD*, pages 426–439 426–439, May 1987. also available as SRC-TR-87-30, UMIACS-TR-86-27, CS-TR-1781.

[Fuk90] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990. 2nd Edition.

[Gae95] V. Gaede. Optimal redundancy in spatial database systems. In *Proc. 4th Int. Symp. on Spatial Databases (SSD'95)*, pages 96–116, 1995.

[Gal91] D. Le Gall. Mpeg: a video compression standard for multimedia applications. *Comm. of ACM (CACM)*, 34(4):46–58, April 1991.

[Gar82] I. Gargantini. An effective way to represent quadtrees. *Comm. of ACM (CACM)*, 25(12):905–910, December 1982.

[GBYS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text: Pat trees and pat arrays. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval*. Prentice Hall, 1992.

[GG95] Volker Gaede and Oliver Guenther. Survey on multidimensional access methods. Technical Report ISS-16, Institut fuer Wirtschaftsinformatik, Humboldt-Universitaet zu Berlin, August 1995.

[GGMT94] Louis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The effectiveness of gloss for the text database discovery problem. *ACM SIGMOD*, pages 126–137, May 1994.

[GK95] Dina Q. Goldin and Paris C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. *Int. Conf. on Principles and Practice of Constraint Programming (CP95)*, September 1995.

[GR94] V. Gaede and W.-F. Riekert. Spatial access methods and query processing in the object-oriented GIS GODOT. In *Proc. of the AGDM'94 Workshop*, pages 40–52, Delft, The Netherlands, 1994. Netherlands Geodetic Commission.

[GR95] Venkat N. Gudivada and Vijay V. Raghavan. Content-based image retrieval systems. *IEEE Computer*, 28(9):18–22, September 1995.

[GT87] G.H. Gonnet and F.W. Tompa. Mind your grammar: a new approach to modelling text. *Proc. of the Thirteenth Int. Conf. on Very Large Data Bases*, pages 339–346, September 1987.

[Gun86] O. Gunther. The cell tree: an index for geometric data. Memorandum No. UCB/ERL M86/89, Univ. of California, Berkeley, December 1986.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, June 1984.

[GVL89] G. H. Golub and C. F. Van-Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1989. 2nd edition.

[Har71] M.C. Harrison. Implementation of the substring test by hashing. *CACM*, 14(12):777–779, December 1971.

[Har94] D. Harman. The second text retrieval conference (trec-2). Special Publication 500-215, National Institute of Standards and Technology, Gaithersburg, MD., 1994.

[Has81] R.L. Haskin. Special-purpose processors for text retrieval. *Database Engineering*, 4(1):16–29, September 1981.

[HD80] P.A.V. Hall and G.R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, December 1980.

[HH83] R.L. Haskin and L.A. Hollaar. Operational characteristics of a hardware-based pattern matcher. *ACM TODS*, 8(1):15–40, March 1983.

[HN83] K. Hinrichs and J. Nievergelt. The grid file: a data structure to support proximity queries on spatial objects. *Proc. of the WG'83 (Intern. Workshop on Graph Theoretic Concepts in Computer Science)*, pages 100–113, 1983.

[Hol79] L.A. Hollaar. Text retrieval computers. *IEEE Computer Magazine*, 12(3):40–50, March 1979.

[Hor86] Berthold Horn. *Robot Vision*. MIT Press, Cambridge, Mass., 1986.

[HS79] G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. on PAMI*, PAMI-1(2):145–153, April 1979.

[HS91] Andrew Hume and Daniel Sunday. Fast string searching. *Software - Practice and Experience*, 21(11):1221–1248, November 1991.

[HS95] M. Houtsma and A. Swami. Set-oriented mining for association rules. In *Proceedings of IEEE Data Engineering Conference*, March 1995. Also appeared as IBM Research Report RJ 9567.

[HSW88] A. Hutflesz, H.-W. Six, and P. Widmayer. Twin grid files: Space optimizing access schemes. *Proc. of ACM SIGMOD*, pages 183–190, June 1988.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, Mass., 1979.

[Jag90a] H.V. Jagadish. Linear clustering of objects with multiple attributes. *ACM SIGMOD Conf.*, pages 332–342, May 1990.

[Jag90b] H.V. Jagadish. Spatial search with polyhedra. *Proc. Sixth IEEE Int'l Conf. on Data Engineering*, February 1990.

[Jag91] H.V. Jagadish. A retrieval technique for similar shapes. *Proc. ACM SIGMOD Conf.*, pages 208–217, May 1991.

[JMM95] H.V. Jagadish, Alberto O. Mendelzon, and Tova Milo. Similarity-based queries. *Proc. ACM SIGACT-SIGMOD-SIGART PODS*, pages 36–45, May 1995.

[JN92] Ramesh Jain and Wayne Niblack. NSF Workshop on Visual Information Management, February 1992.

[JS89] Theodore Johnson and Dennis Shasha. Utilization of b-trees with inserts, deletes and modifies. *Proc. of ACM SIGACT-SIGMOD-SIGART PODS*, pages 235–246, March 1989.

[Jur92] Ronald K. Jurgen. Digital video. *IEEE Spectrum*, 29(3):24–30, March 1992.

[KF93] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. *Second Int. Conf. on Information and Knowledge Management (CIKM)*, November 1993.

[KF94] Ibrahim Kamel and Christos Faloutsos. Hilbert R-tree: An Improved R-tree Using Fractals. In *Proceedings of VLDB Conference,*, pages 500–509, Santiago, Chile, September 1994.

[Kim88] R.E. Kimbrell. Searching for text? send an n-gram! *Byte*, 13(5):297–312, May 1988.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comput*, 6(2):323–350, June 1977.

[Kno75] G.D. Knott. Hashing functions. *Computer Journal*, 18(3):265–278, 1975.

[Knu73] D.E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, Reading, Mass, 1973.

[KR87] R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, March 1987.

[KS91] Henry F. Korth and Abraham Silberschatz. *Database System Concepts*. McGraw Hill, 1991.

[KSF+96] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot Siegel, and Zenon Protopapas. Fast nearest-neighbor search in medical image databases. *Conf. on Very Large Data Bases (VLDB)*, September 1996. Also available as Univ. of Maryland tech. report: CS-TR-3613, ISR-TR-96-13.

[KTF95] Anil Kumar, Vassilis J. Tsotras, and Christos Faloutsos. Access methods for bi-temporal databases. *Int. Workshop on Temporal Databases*, September 1995.

[KW78] Joseph B. Kruskal and Myron Wish. *Multidimensional scaling*. SAGE publications, Beverly Hills, 1978.

[Lar78] P. Larson. Dynamic hashing. *BIT*, 18:184–201, 1978.

[Lar82] P.A. Larson. Performance analysis of linear hashing with partial expansions. *ACM TODS*, 7(4):566–587, December 1982.

[Lar85] P.A. Larson. Hash files: Some recent developments. In *Proc. of the First Intern. Conference on Supercomputing Systems*, pages 671–679, St. Petersburg, Florida, December 1985.

[Lar88] P.-A. Larson. Dynamic hash tables. *Comm. of ACM (CACM)*, 31(4):446–457, April 1988.

[LeB92] Blake LeBaron. Nonlinear forecasts for the s\&p stock index. In M. Castagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting*, pages 381–393. Addison Wesley, 1992. Proc. Vol. XII.

[Les78] M.E. Lesk. *Some Applications of Inverted Indexes on the UNIX System*. Bell Laboratories, Murray Hill, New Jersey, 1978.

[Lit80] W. Litwin. Linear hashing: a new tool for file and table addressing. In *Proc. 6th International Conference on VLDB*, pages 212–223, Montreal, October 1980.

[LL89] D.L. Lee and C.-W. Leng. Partitioned signature file: Designs and performance evaluation. *ACM Trans. on Information Systems (TOIS)*, 7(2):158–180, April 1989.

[LR94] Ming-Ling Lo and Chinya V. Ravishankar. Spatial joins using seeded trees. *ACM SIGMOD*, pages 209–220, May 1994.

[LS90] David B. Lomet and Betty Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM TODS*, 15(4):625–658, December 1990.

[LSM91] Xia Lin, Dagobert Soergel, and Gary Marchionini. A self-organizing semantic map for information retrieval. *Proc. of ACM SIGIR*, pages 262–269, October 1991.

[Lum70] V.Y. Lum. Multi-attribute retrieval with combined indexes. *CACM*, 13(11):660–665, November 1970.

[LW75] R. Lowerance and R.A. Wagner. An extension of the string-to-string correction problem. *JACM*, 22(2):3–14, April 1975.

[LW89] Carl E. Langenhop and William E. Wright. A model of the dynamic behavior of b-trees. *Acta Informatica*, 27:41–59, 1989.

[Man77] B. Mandelbrot. *Fractal Geometry of Nature*. W.H. Freeman, New York, 1977.

[Mar79] G.N.N. Martin. Spiral storage: Incrementally augmentable hash addressed storage. Theory of Computation, Report No. 27, Univ. of Warwick, Coventry, England, March 1979.

[McI82] M.D. McIlroy. Development of a spelling list. *IEEE Trans. on Communications*, COM-30(1):91–99, January 1982.

[MJFS96] Bongki Moon, H.V. Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of hilbert space-filling curve. Technical Report CS-TR-3611, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 1996.

[ML86] Lothar M. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for distributed queries. *Proc. of 12th Int. Conf. on Very Large Data Bases (VLDB)*, August 1986.

[Moo49] C. Mooers. Application of random codes to the gathering of statistical information. Bulletin 31, Zator Co, Cambridge, Mass, 1949. based on M.S. thesis, MIT, January 1948.

[Mor68] Donald R. Morrison. Patricia - practical algorithm to retrieve information coded in alphanumeric. *Journal of ACM (JACM)*, 15(4):514–534, October 1968.

[MRT91] Carlo Meghini, Fausto Rabitti, and Constantino Thanos. Conceptual modeling of multimedia documents. *IEEE Computer*, 24(10):23–30, October 1991.

[Mur83] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[MW94] Udi Manber and Sun Wu. Glimpse: a tool to search through entire file systems. *Proc. of USENIX Techn. Conf.*, 1994. Also available as TR 93-94, Dept. of Comp. Sc., Univ. of Arizona, Tucson, or through anonymous ftp (ftp://cs.arizona.edu/glimpse/glimpse.ps.Z).

[NBE+93] Wayne Niblack, Ron Barber, Will Equitz, Myron Flickner, Eduardo Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. The qbic project: Querying images by content using color, texture and shape. *SPIE 1993 Intl. Symposium on Electronic Imaging: Science and Technology, Conf. 1908, Storage and Retrieval for Image and Video Databases*, February 1993. Also available as IBM Research Report RJ 9203 (81511), Feb. 1, 1993, Computer Science.

[NC91] A. Desai Narasimhalu and Stavros Christodoulakis. Multimedia information systems: the unfolding of a reality. *IEEE Computer*, 24(10):6–8, October 1991.

[NHS84] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM TODS*, 9(1):38–71, March 1984.

[Nof86] P.J. Nofel. 40 million hits on optical disk. *Modern Office Technology*, pages 84–88, March 1986.

[ODL93] Katia Obraczka, Peter B. Danzig, and Shih-Hao Li. Internet resourse discovery services. *IEEE Computer*, September 1993.

[OM84] J.A. Orenstein and T.H. Merrett. A class of data structures for associative searching. *Proc. of SIGACT-SIGMOD*, pages 181–190, April 1984.

[OM88] J.A. Orenstein and F.A. Manola. Probe spatial data modeling and query processing in an image database application. *IEEE Trans. on Software Engineering*, 14(5):611–629, May 1988.

[Ore86] J. Orenstein. Spatial query processing in an object-oriented database system. *Proc. ACM SIGMOD*, pages 326–336, May 1986.

[Ore89] J.A. Orenstein. Redundancy in spatial databases. *Proc. of ACM SIGMOD Conf.*, May 1989.

[Ore90] J.A. Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. *Proc. of ACM SIGMOD Conf.*, pages 343–352, 1990.

[OS75] Alan Victor Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, N.J., 1975.

[OS95] Virginia E. Ogle and Michael Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28(9):40–48, September 1995.

[Pet80] J.L. Peterson. Computer programs for detecting and correcting spelling errors. *CACM*, 23(12):676–687, December 1980.

[PF94] Euripides G.M. Petrakis and Christos Faloutsos. Similarity searching in large image databases, 1994. submitted for publication. Also available as technical report at MUSIC with # TR-01-94.

[PF96] Euripides G.M. Petrakis and Christos Faloutsos. Similarity searching in medical image databases. *IEEE Trans. on Knowledge and Data Engineering (TDKE)*, 1996. To appear. Also available as technical report at MUSIC with # TR-01-94, UMIACS-TR-94-134, CS-TR-3388.

[PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[PO95] E. G.M. Petrakis and S. C. Orphanoudakis. A Generalized Approach for Image Indexing and Retrieval Based Upon 2-D Strings. In S.-K. Chang, E. Jungert, and G. Tortora, editors, *Intelligent Image Database Systems - Spatial Reasoning, Image Indexing and Retrieval using Symbolic Projections*. World Scientific Pub. Co., 1995. To be publised. Also available as FORTH-ICS/TR-103.

[Pri84] Joseph Price. The optical disk pilot project at the library of congress. *Videodisc and Optical Disk*, 4(6):424–432, November 1984.

[PSTW93] B. Pagel, H. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance. *Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 214–221, May 1993.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992. 2nd Edition.

[Ras92] Edie Rasmussen. Clustering algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 419–442. Prentice Hall, 1992.

[RBC+92] Mary Beth Ruskai, Gregory Beylkin, Ronald Coifman, Ingrid Daubechies, Stephane Mallat, Yves Meyer, and Louise Raphael. *Wavelets and Their Applications*. Jones and Bartlett Publishers, Boston, MA, 1992.

[RF91] Yi Rong and Christos Faloutsos. Analysis of the clustering property of peano curves. Techn. Report CS-TR-2792, UMIACS-TR-91-151, Univ. of Maryland, December 1991.

[Riv76] R.L. Rivest. Partial match retrieval algorithms. *SIAM J. Comput*, 5(1):19–50, March 1976.

[RJ93] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[RKV95] Nick Roussopoulos, Steve Kelley, and F. Vincent. Nearest Neighbor Queries. *Proc. of ACM-SIGMOD*, pages 71–79, May 1995.

[RL85] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed R-trees. *Proc. ACM SIGMOD*, May 1985.

[RMS92] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neural Computation and Self-Organizing Maps*. Addison Wesley, Reading, MA, 1992.

[Rob79] C.S. Roberts. Partial-match retrieval via the method of superimposed codes. *Proc. IEEE*, 67(12):1624–1642, December 1979.

[Rob81] J.T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. *Proc. ACM SIGMOD*, pages 10–18, 1981.

[Roc71] J.J. Rocchio. Performance indices for document retrieval. In G. Salton, editor, *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1971. Chapter 3.

[RS92] Fausto Rabitti and Pascuale Savino. An information retrieval approach for image databases. In *VLDB Conf. Proceedings*, pages 574–584, Vancouver, BC, Canada, August 1992.

[RV91] Oliver Rioul and Martin Vetterli. Wavelets and signal processing. *IEEE SP Magazine*, pages 14–38, October 1991.

[Sal71a] G. Salton. Relevance feedback and the optimization of retrieval effectiveness. In G. Salton, editor, *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1971. Chapter 15.

[Sal71b] G. Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1971.

[Sch91] Manfred Schroeder. *Fractals, Chaos, Power Laws: Minutes From an Infinite Paradise*. W.H. Freeman and Company, New York, 1991.

[SD76] D.G. Severance and R.A. Duhne. A practitioner's guide to addressing algorithms. *CACM*, 19(6):314–326, 1976.

[SDKR87] R. Sacks-Davis, A. Kent, and K. Ramamohanarao. Multikey access methods based on superimposed coding techniques. *ACM Trans. on Database Systems (TODS)*, 12(4):655–696, December 1987.

[SDR83] R. Sacks-Davis and K. Ramamohanarao. A two level superimposed coding scheme for partial match retrieval. *Information Systems*, 8(4):273–280, 1983.

[SFW83] G. Salton, E.A. Fox, and H. Wu. Extended boolean information retrieval. *CACM*, 26(11):1022–1036, November 1983.

[Sha88] C.A. Shaffer. A formula for computing the number of quadtree node fragments created by a shift. *Pattern Recognition Letters*, 7(1):45–49, January 1988.

[Sig93] Karl Sigmund. *Games of Life: Explorations in Ecology, Evolution and Behaviour*. Oxford University Press, 1993.

[SK83] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparisons*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1983.

[SK86] C. Stanfill and B. Kahle. Parallel free-text search on the connection machine system. *CACM*, 29(12):1229–1239, December 1986.

[SK90] Bernhard Seeger and Hans-Peter Kriegel. The buddy-tree: an efficient and robust access method for spatial database systems. *Proc. of VLDB*, pages 590–601, August 1990.

[SL76] D.G. Severance and G.M. Lohman. Differential files: Their application to the maintenance of large databases. *ACM TODS*, 1(3):256–267, September 1976.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[SOF+92] G.A. Story, L. O'Gorman, D.S. Fox, L.L. Schaper, and H.V. Jagadish. The rightpages: an electronic library for alerting and browsing. *IEEE Computer*, 25(9):17–26, September 1992.

[SS88] R. Stam and Richard Snodgrass. A bibliography on temporal databases. *IEEE Bulletin on Data Engineering*, 11(4), December 1988.

[SSH86] M. Stonebraker, T. Sellis, and E. Hanson. Rule indexing implementations in database systems. In *Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC, April 1986.

[SSN87] C.A. Shaffer, H. Samet, and R.C. Nelson. Quilt: a geographic information system based on quadtrees. Technical Report CS-TR-1885.1, Univ. of Maryland, Dept. of Computer Science, July 1987. to appear in the International Journal of Geographic Information Systems.

[ST84] J. Stubbs and F.W. Tompa. Waterloo and the new oxford english dictionary project. In *Proc. of the Twentieth Annual Conference on Editorial Problems*, Toronto, Ontario, November 1984. in press.

[Sta80] T.A. Standish. *Data Structure Techniques*. Addison Wesley, 1980.

[Sti60] S. Stiassny. Mathematical analysis of various superimposed coding methods. *American Documentation*, 11(2):155–169, February 1960.

[Str80] Gilbert Strang. *Linear Algebra and its Applications.* Academic Press, 1980. 2nd edition.

[Sun90] D.M. Sunday. A very fast substring search algorithm. *Comm. of ACM (CACM)*, 33(8):132–142, August 1990.

[SW78] G. Salton and A. Wong. Generation and search of clustered files. *ACM TODS*, 3(4):321–346, December 1978.

[TC83] D. Tsichritzis and S. Christodoulakis. Message files. *ACM Trans. on Office Information Systems*, 1(1):88–98, January 1983.

[TGMS94] Anthony Tomasic, Hector Garcia-Molina, and Kurt Shoens. Incremental updates of inverted lists for text document retrieval. *ACM SIGMOD*, pages 289–300, May 1994.

[Ton90] Howell Tong. *Non-Linear Time Series: a Dynamical System Approach.* Clarendon Press, Oxford, 1990.

[TSW+85] G.R. Thoma, S. Suthasinekul, F.A. Walker, J. Cookson, and M. Rashidian. A prototype system for the electronic storage and retrieval of document images. *ACM TOOIS*, 3(3), July 1985.

[Vas93] Dimitris Vassiliadis. The input-state space approach to the prediction of auroral geomagnetic activity from solar wind variables. *Int. Workshop on Applications of Artificial Intelligence in Solar Terrestrial Physics*, September 1993.

[VM] Brani Vidakovic and Peter Mueller. *Wavelets for Kids.* Duke University, Durham, NC. ftp://ftp.isds.duke.edu/pub/Users/brani/papers/.

[VR79] C.J. Van-Rijsbergen. *Information Retrieval.* Butterworths, London, England, 1979. 2nd edition.

[Wal91] Gregory K. Wallace. The jpeg still picture compression standard. *CACM*, 34(4):31–44, April 1991.

[WG94] Andreas S. Weigend and Neil A. Gerschenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison Wesley, 1994.

[Whi81] M. White. *N-Trees: Large Ordered Indexes for Multi-Dimensional Space.* Application Mathematics Research Staff, Statistical Research Division, U.S. Bureau of the Census, December 1981.

[WM92] Sun Wu and Udi Manber. Text searching allowing errors. *Comm. of ACM (CACM)*, 35(10):83–91, October 1992.

[Wol91] Stephen Wolfram. *Mathematica.* Addison Wesley, 1991. Second Edition.

[WS93] Kuansan Wang and Shihab Shamma. Spectral shape analysis in the central auditory system. *NNSP*, September 1993.

[WTK86] A. Witkin, D. Terzopoulos, and M. Kaas. Signal matching through scale space. *Proc. am. Assoc. Artif. Intel.*, pages 714–719, 1986.

[WZ96] Hugh Williams and Justin Zobel. Indexing nucleotide databases for fast query evaluation. *Proc. of 5-Th Intl. Conf. on Extending Database Technology (EDBT)*, pages 275–288, March 1996. Eds. P. Apers, M. Bouzeghoub, G. Gardarin.

[Yao78] A. C. Yao. On random 2,3 trees. *Acta Informatica*, 9:159–170, 1978.

[YGM94] Tak W. Yan and Hector Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *ACM TODS*, 19(2):332–364, June 1994.

[YMD85] N. Yankelovich, N. Meyrowitz, and N. Van Dam. Reading and writing the electronic book. *IEEE Computer*, pages 15–30, October 1985.

[Zah71] C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. on Computers*, C-20(1):68–86, January 1971.

[ZC77] A.L. Zobrist and F.R. Carlson. Detection of combined occurrences. *CACM*, 20(1):31–35, January 1977.

[Zip49] G.K. Zipf. *Human Behavior and Principle of Least Effort: an Introduction to Human Ecology.* Addison Wesley, Cambridge, Massachusetts, 1949.

[ZMSD92] Justin Zobel, Alistair Moffat, and Ron Sacks-Davis. An efficient indexing technique for full-text database systems. *VLDB*, pages 352–362, August 1992.

[ZRT91] P. Zezula, F. Rabitti, and P. Tiberio. Dynamic partitioning of signature files. *ACM TOIS*, 9(4):336–369, October 1991.

INDEX