Hierarchies of Indices for Text Searching

Ricardo Baeza-Yates

Departamento de Ciencias de la Computación
Universidad de Chile
Santiago, Chile

Eduardo Fernandes Barbosa

Departamento de Engenharia Eletrônica
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil

Nivio Ziviani

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil

Abstract

The objective of this paper is to present an efficient implementation of a recently known index for text databases presented in the literature, when the database is stored on secondary storage devices such as magnetic or optical disks. The implementation is built on top of a new and simple index for texts called PAT array (also called suffix array). Considering that text searching in a large database spends most of the time accessing external storage devices, we propose additional indices structures and searching algorithms for PAT arrays that reduce the number of disk accesses.

We present two index structures: a two-level hierarchy model that uses the main memory and one level of external storage (magnetic or optical devices) and a three-level hierarchy model that uses the main memory and two levels of external storage (magnetic and optical devices). Performance improvement is achieved in both models by storing most of higher index levels in faster memories, thus reducing accesses in the slowest devices in the hierarchy. Analytical results and performance gains are presented for both models. Further improvements for text searching with these models (such as index compression) are also proposed.

KEY–WORDS: Text searching, PAT arrays, suffix arrays, magnetic disks, read-only optical disks, CD-ROM, index hierarchy, memory hierarchy.

1 Introduction

The preponderant use of the computer for typesetting in the publishing industry and the continued decline in the cost of mass storage devices – both magnetic and optical media – make textual database one of the fastest growing category of databases. For static databases it is worthwhile to preprocess the database and build an index to speed up the searching time. A new and simple type of index is the PAT array (Gonnet and Baeza-Yates, 1991). A PAT array is a compact representation of a digital tree called PAT tree, because it stores only the external nodes of the tree. A PAT tree is a Patricia tree (Morrison, 1968) built on all indexing points (called semi-infinite strings or sistrings for short) of a text database. The PAT tree was originally described by Gonnet (1983) and later used in conjunction with the computerization of the Oxford English Dictionary (Gonnet, 1987).

The most important complexity measures for preprocessed text databases are (i) the time required to build the index, (ii) the time required to search for a particular query and (iii) the extra space used by the index. Building a PAT array is similar to sorting variable length records, at a cost of $O(n \log n)$, where n indicates the size of the text database, either characters or number of indexing points. Searching in a PAT array takes at most $2\,m \log_2 n$ character comparisons and at most $4 \log_2 n$ disk accesses, where m is the length of a given query. The extra space used by a PAT array is only one pointer per indexing point (Gonnet and Baeza-Yates, 1991).

The main objective in the design of any memory system is to obtain adequate storage capacity, with good level of performance at the lowest possible cost. One of the solutions to achieve this objective is to distribute the information in the available storage devices and conceive data structures and algorithms that reduce the number of seeks in the slowest devices. The existence of different memory devices with different cost-performance ratios gave rise to the concept of *memory hierarchy*. The differences in cost-performance for different memory devices can be sumarized as follows: larger capacity devices are associated to low cost and slow access time memories, while smaller capacity devices are associated to faster access and higher cost memories. For instance, cache memories are fast and expensive devices with small capacity, while CD-ROM disks are very slow and very cheap devices with large capacity.

In the last few years new technologies have emerged and the optical disk, specifically the read-only disk known as CD-ROM (Compact Disk Read-Only Memory), has proved to be the best choice for low cost large textual database distribution. This is due to its large storage capacity (up to 600 megabytes), low cost to the end user (it can be as low as two dollars per unit for large scale production) and good physical and logical integrity. However, seeking in the CD-ROM is a very high time-consuming task when compared with magnetic disks, as CD-ROM disks may be 15 to 30 times slower than the magnetic disk. Therefore, a naive implementation of the PAT array index for read-only optical disks might result in a poor performance from a practical point of view.

The objective of this paper is to present an efficient implementation of PAT arrays for external storage devices holding large textual databases. Due to the widespread use of magnetic and optical media we concentrate our analysis on these two storage media. We propose additional indices structures and searching algorithms that improve searching time by reducing the number of disk accesses. We show how to distribute the PAT array in the memory hierarchy so that most of the higher levels of the index structure is stored in faster memories and only a few index levels are left for the slowest device in the hierarchy. This approach effectively reduces the number of disk accesses needed to answer a query, thus improving the searching performance. A previous version of this paper was presented in Baeza-Yates, Barbosa and Ziviani, 1993.

In the Section 2 we briefly describe the diversity of external storage device performances with emphasis in the characteristics of magnetic and optical disks. In Section 3 we present the structure of the PAT array and exemplify the indexing and retrieval operations for a very simple text database. In Section 4 we present the two-level hierarchy model which uses the main memory as the first level and magnetic or optical disk as the second level. In Section 5 we present the three-level hierarchy model, which distributes the indices among main memory, magnetic disk and optical disk. The cost analysis, the feasible performance gain and advantages that can be achieved by using these models are also presented.

2 Magnetic Versus Optical Disks

The existence of various technologies for information storage with different cost-performance ratios raised the need for a design strategy that could provide adequate storage capacity with good level of performance at the lowest cost possible. The concept that supports this strategy, known as *memory hierarchy*, consists of distributing the information over a variety of different memory devices with very different physical characteristics, so that the entire memory system can achieve the design objectives.

In a typical computer system we can think in five levels of storage, as shown in Table 1. External storage devices, such as magnetic and optical disks, normally constitute the last levels in the memory hierarchy, where the database of interest is normally stored. The space and time trade-offs has to be considered in the design of efficient information retrieval systems. Some values in Table 1 could be different for small computer systems, and some figures may change as a consequence of technological improvements. In a personal computer, for example, the main memory size is normally in the range of 8 to 32 megabytes, and the actual magnetic disks are usually smaller than 500 megabytes. Magnetic disk capacity is expected to double each 2 years while seek time is halfed only at each 10 years, and the typical rotational delay for the magnetic disk did not change in the last decade (Katz, Gibson and Patterson, 1989).

Level	1	2	3	4	5
Type	CPU Registers	Cache Memory	Main Memory	Magnetic Disk	Optical Disk (CD-ROM)
Size	$< 1KB$	$< 512KB$	$< 256MB$	$> 256MB$	$600MB$
Access Time	1×10^{-5} mS	2×10^{-5} mS	1×10^{-4} mS	20 mS	≈ 400 mS
Cost(U$/MB)	–	≈ 300	≈ 30	≈ 1.50	< 0.01

Table 1: Typical performance and cost values for memory devices in a computer system

The access time is the main parameter of interest when we analyse the retrieval performance in the last levels of the memory hierarchy. The access time, t_a is:

$$t_a = t_s + t_r + t_x \tag{1}$$

where t_s is the seek time, the delay incurred from the instant that the device received the signal to access a given disk sector until the reading mechanism is placed in the correct track; t_r is the average time needed for the desired disk sector to reach the reading mechanism, called rotational delay (equal to half the time needed for one disk rotation); and t_x is the time needed to transfer the data from the disk to the next level of the memory hierarchy, in this case, the main memory. These three time delays are the main components of the access time in any disk system, regardless of technological aspects.

For analytical purposes, we selected three levels in the memory hierarchy to build up our model: the main memory, the magnetic disk and the read-only optical disk – CD-ROM. This choice was based on the following: firstly, the difference of access time from one level to the next should be at least one order of magnitude. Based on this criteria, the erasable optical disk, specifically the magnetic-optical, was not considered in our analysis because this device is physically and logically very similar to the magnetic disk. Thus, the analysis that will be carried out for the magnetic disk can be easily adapted to the erasable optical disk. Secondly, we do not consider the write-once read-many (WORM) disk because it is not of practical use for large textual data base distribution. In addition, the WORM disk has its retrieval performance strongly affected by the existing file system and by many other factors such as efficiency in space utilization and data reorganization during updatings.

In the following we concentrate our discussion on the main differences between magnetic disks and CD-ROM disks, and how their characteristics affect the retrieval performance.

Characteristic	Magnetic Disk	Optical Disk (CD-ROM)
Physical format	CAV (Constant Angular Velocity)	CLV (Constant Linear Velocity)
Addressing mode	Cylinder/Track/Sector	Sector number
Info Structure	Dynamic	Static
Seek cost function	Weakly position dependent	Highly position dependent
Track capacity	Fixed (e.g. 32 sectors)	9 to 20 sectors (Variable)
Sector size	0.5 to 4 KB	2 KB (fixed)
Track-to-track time	3 mS	1 mS
Average seek-time	15 mS	400 mS
Average latency	8.3 mS	60 to 140 mS
Transfer rate	> 1 MB/S	0.15 or 0.30 MB/S

Table 2: Characteristics of read-only optical disks and magnetic disks

Table 2 sumarizes the main differences between CD-ROM disks and magnetic disks. We emphasize the following: (i) the CD-ROM disk is a read-only medium and, as such, the structure of the information is static; (ii) the efficiency of data access in the CD-ROM disk is affected by its location on the disk and by the sequence in which it is retrieved and (iii) due to constant linear velocity (CLV) the tracks of CD-ROM have variable capacity and rotational latency depends on disk position.

The CD-ROM track has a spiral pattern and an access to distant tracks requires more time due to the need of optical head displacement and changes in disk rotation. The capability of accessing nearby tracks with no displacement of the optical mechanism is called *span* and the number of tracks which can be accessed in this way is called *span size*. In actual CD-ROM drives span size is up to $\pm$ 30 tracks or 60 tracks. The access of data located within span boundaries is efficient and seek time is reduced to about 1 millisecond per additional track. When accessing data located outside span boundaries, seek time requires from 200 to 600 milliseconds due to optical head displacement, against approximately 10 to 20 milliseconds for a typical magnetic disk.

In the CD-ROM the rotational latency, file size, file structure, and file allocation strongly affects the retrieval performance. The influence of the skewed track capacity distribution and rotational latency variations leads to tradeoffs between the number of seeks and rotational delays which are the main components for the total retrieval costs in CD-ROM disks (Barbosa and Ziviani, 1992).

The high retrieval costs inherent to CD-ROM technology reinforces the need for designing algorithms and data structures that compensate its low performance. A 300 megabytes text file, for example, with $n = 50$ million index points would require, in the worst case, $A_n = 4 \times \log\ n = 102$ disk accesses. If a magnetic medium is used, seek time (t_s) has an average value of 0.015 seconds, and this search could be completed in (at most) $102 \times t_s = 1.53$ seconds. If the medium is a CD-ROM disk, where t_s has an average value of 0.40 seconds the same search would take about 102×0.40 seconds $= 40.80$ seconds, which is 26 times greater. This example shows that any improvement in the retrieval costs will result in great benefit to the end user, when the data base is stored in CD-ROM disks.

If we consider that more complex queries may require a very large number of disk accesses, text retrieval in large database stored in magnetic disks will also present better performance if we reduce the number of disks accesses to answer the query. By complex queries we mean proximity searching or logical searching, which may require many hundreds or even thousands of disk accesses before finding the desired answer.

Some common solutions for reducing disk accesses in CD-ROM applications are: (i) store the main index structure in main memory; (ii) use of proximal storage technique, by storing correlated data in near place on disk and (iii) employ a redundancy technique, duplicating data on disk region where they are frequently required for retrieval.

When the PAT array is associated to a large text database, none of these solutions are of practical application. The reasons are: (i) if the main memory is used for full index storage, the main database size will be bounded by very narrow values of text size. In a PAT array structure, the index size is about 60% of text size (Gonnet, Baeza-Yates and Snider, 1992); (ii) in a random search scheme, there is no deterministic way of finding out what is the correlated data when a random query is processed and (iii) duplication of data to obtain cost reduction in all queries is a difficult planning task and it clearly restrains the maximum text file size that could be stored on the disk.

3 PAT Arrays

The traditional model of text divides a textual database in a *set of documents*. A list of significant words, called *keywords*, are extracted from the text and assigned to each document. Consequently, queries are restricted to pre-selected keywords.

A new approach for textual databases sees the text as one long string (Gonnet and Baeza-

Yates, 1991; Manber and Myers, 1990). Each position in the text is called a *semi-infinite string* (or *sistring* for short). A sistring is defined by a starting position and extends to the right as far as needed or to the end of the text. The database may be viewed as having an infinite number of null characters at its right end. Sistrings are compared lexicographically character by character, and so any two strings in different positions do not compare equal.

Depending on the character sequences that are likely to be sistrings for searching, the user must define which positions of the text will be indexed. An *index point* is a position in the text that must be indexed. In large databases, not all character sequences are indexed, just the alphabetical symbols that are preceded by a blank symbol.

Figure 1 illustrates an example of a text database with nine index points. Each index point corresponds to the address of a sistring. Figure 2 shows the nine sistrings in lexicographical ascending order.

This	text	is	an	example	of	a	textual	database
1	6	11	14	17	25	28	30	38

Figure 1: A simple text database with nine index points

28	a textual database
14	an example of a textual database
38	database
17	example of a textual database
11	is an example of a textual database
25	of a database
6	text is an example of a textual database
30	textual database
1	This text is an example of a textual database

Figure 2: Database ordered by sistrings

Any search structure that allows for range searches can be used to search on the set of sistrings. However, the most suitable structures are digital trees, in particular Patricia trees (Morrison, 1968). A Patricia tree built on all the sistrings of a text database was suggested by Gonnet (1983), and later on called PAT trees (Gonnet and Baeza-Yates, 1991).

In the PAT tree structure (1) the search is done over the tree while scanning the bits of the search string (there is no need to compare the text during the search); (2) the whole set of sistrings answering a query is contained in a single subtree and hence the searching time is independent of the size of the answer. Every subtree of the PAT tree contains, by construction, all the sistrings with a given prefix. Prefix searching in a PAT tree ends when the prefix is exhausted. At that point all the answer is available in a single subtree.

PAT trees are very efficient because only $O(\log n)$ bit inspections are necessary to obtain the whole set of sistrings answering a query. However, the PAT tree needs at least two pointers for every internal node and one pointer to every external node, which means $12n$ bytes for n index points.

A simple, efficient and elegant solution to this problem is to store only the external nodes of the PAT tree. Thus, the whole index turns into a single array of external nodes ordered lexicographically by sistrings. This single array was called PAT *array* by Gonnet during the computerization of the Oxford English Dictionary. This idea was independently discovered by Manber and Myers (1990), where it is called *suffix array*.

Figure 3 shows the PAT array for the sample text database presented in the Figure 1. The size of the index is now reduced to $4n$ bytes, where n is the number of index points. This represents a significant economy in space at the cost of a factor of $\log_2 n$ (Gonnet, Baeza-Yates and Snider, 1992). In this case prefix searching can be implemented by doing an indirect binary search over the PAT array with the results of the comparison being less than, equal to or greater than. To search for the prefix *tex* in the PAT array presented in Figure 3 we perform an indirect binary search over the array and obtain [7, 8] as answer (position 7 of the PAT array points to the sistring beginning at the position 6 in the text, and position 8 of the PAT array points to the sistring beginning at position 30 in the text). The size of the answer in this case is 2, which is the size of the array interval.

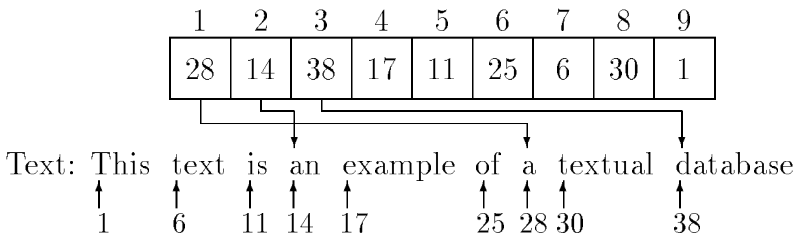

Figure 3: PAT array

4 Two Level Hierarchy Model

In this Section we consider a two-level memory hierarchy model composed of the main memory and a secondary storage device – either magnetic or optical disk. Because the main memory is more than 200,000 times faster than the faster secondary storage device (the magnetic disk) and more than 4,000,000 times faster than the slowest device (the CD-ROM disk) we might consider the cache memory and the internal processor registers as part of the first level of our hierarchy model, with no loss of accuracy.

General description of the Model

Text searching with the PAT array model is an indirect binary search that uses two files: one file containing the PAT array (pointers to the text) and the other file that is the text itself. We assume that both files reside on the disk. If we consider the searching in a PAT array equivalent to the searching in a binary search tree then the worst case occurs when the search starts at the root of the tree and goes down to the leaves. In this case, for n index points in the PAT array, a prefix or range searching will take, in the worst case, $A_n = 4 \log n$ disk accesses. Half of these accesses are used to find a left bound and the other half to find the right bound of the answer in the PAT array. Observe that each PAT entry requires 2 accesses during the searching process: one to read the PAT entry itself, and the other to read the text in the position pointed to by each PAT entry.

Our solution to improve the performance of the searching process is to divide the PAT array in equal size blocks and move one element of each block (a node from a PAT array subtree) to main memory, together with additional information about the text related to (pointed by) the element selected from each block. Placing elements of the PAT array in main memory is equivalent to store part of the search tree – a PAT subtree – in the faster storage device. Due to the additional information about the text in the upper index structure, the binary search can be made directly, because there is no need to access the disk while the search is being done in the main memory. As a result, most of the searching work is done in main memory, thus reducing by half the number of accesses in the slowest devices (disks).

Our hierarchy model is based upon an additional data structure and a modified searching algorithm. The data structure is a set of PAT array entries where each entry carries a fixed amount of characters from the text. This structure can be considered a reduced representation of the PAT array and the text file. We call this index *Short*-PAT-*Array*, or SPAT array. The SPAT array is a sorted array of strings where each one contains the first l_s characters of a sistring. The PAT array can be seen as a set of equal size blocks or subtrees: for each block we store in the SPAT array the first l_s characters of the sistring pointed to by the last entry in each PAT array block.

Figure 4 illustrates a SPAT array index for the PAT array of Figure 3. We divide the PAT array in 3 blocks, each one with 3 entries. For each PAT array block we transfer to the SPAT array l_s characters from the sistring pointed to by the last PAT array entry in this block ($l_s = 3$ in this example). As each entry in the SPAT array corresponds to a block in the PAT array with b elements, then SPAT$[i]$ corresponds to the PAT$[b \times i]$ position in the PAT array. Therefore, the pointer from a SPAT array entry to a PAT array block is an *implicit address*.

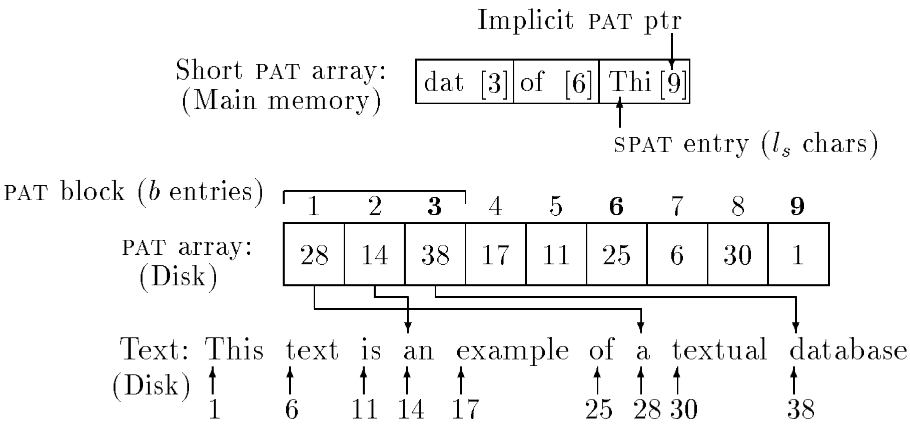

Figure 4: *Short* PAT *array* (SPAT)

The search is done in two phases: first, the SPAT array is transferred to the main memory, where a binary search is done in the reduced index structure, with no disk accesses. This search gives the SPAT entry that points to a small subtree (PAT array block) that contains the desired answer. Then, the PAT array block encountered in the first search phase is moved from disk to memory. Finally, a second binary search is done between memory (PAT subtree containing the answer) and the disk (text file), where the exact bounds of the answer are found. If the answer is a large number of text strings, it may be bounded by at most two SPAT array entries. In this case, a left and right blocks are used in the last phase of the binary searching following the same procedure. By carefully designing this hierarchical model we can make the subtree used in the last binary search sufficiently small, and only a few disk accesses are needed to find the answer.

We now define certain measures:

1. Let M indicate the main memory available to store the SPAT array index, in bytes.
2. Let l_s indicate the number of characters of a SPAT array entry (l_s is selected such that no SPAT entries are equal).
3. Let n indicate the number of index points in the text.
4. Let r indicate the number of elements of the SPAT array ($r = M/l_s$).
5. Let b indicate the number of elements in a PAT array block ($b = n/r = (n \times l_s)/M$)
6. Let B_p indicate the size of each PAT array block, in bytes ($B_p = 4 \times b$).

The algorithm to build the SPAT array structure from an existing PAT array index is:

```
1.  Define the main memory size (M) to be used by the SPAT array;
2.  Select the number of characters for each SPAT array entry (l_s);
3.  Compute the number of elements of the SPAT array:  r = M/l_s;
4.  Divide the PAT array in r blocks of equal size:  B_p = Sizeof(PAT)/r;
5.  For i = 1 to r do
      For each i block of the PAT array
        Copy the first l_s chars of the sistring corresponding to the
        (b×i)^th entry of the PAT array to the i^th entry of the SPAT array;
```

The searching is done in two phases, as follows:

```
1.  Read the SPAT array file to main memory; {Done only once}
2.  Repeat
    2.1.  Find left and right bounds in the SPAT array;{Cost ≈ 0}
    2.2.  Transfer left and right blocks of PAT array from disk to memory;
    2.3.  With the left block do binary search between memory and text file
          {find exact left bound}
    2.4.  With the right block do binary search between memory and text file
          {find exact right bound}
    2.5.  Read the occurrences from text file as desired;
    Until {End of queries}
```

Ideally, the amount of main memory should be as large as possible, but this is not only an expensive solution for very large text files but also will be reflected in low disk space utilization, because the SPAT array file is permanently stored in some disk area, before being transferred to main memory. If we divide the space in main memory (M) by the PAT array size ($n \times 4$) then we have a parameter for memory overhead. As an example, if we allocate a 4 megabytes memory space to accommodate a SPAT array for a text with $n = 50$ million indexing points and each PAT array entry requires 4 bytes, then we have an overhead of only 2.0%. This overhead is still of less significance if we compare the SPAT array index size with the text file size associated to it. Therefore, the space requirement for the SPAT array is very small if compared with the PAT array index requirement.

Performance improvement is achieved with the two-level hierarchy model due to the fact that: (i) a first search phase to find the left and the right bounds in PAT array blocks is performed in main memory with negligible cost; (ii) the second search phase is performed over a much smaller index structure, since it will be done in a small fraction of the PAT array index (a small PAT subtree); and (iii) the exact left and right bounds of occurrences in the text file is performed with half of the number of seeks. This is because the small PAT array block found during the first phase is transferred to main memory and the exact left and right bounds in the text are found through a binary search between *memory and disk* and not between a *disk region* (PAT array) and *another disk region* (text file).

Analytical Results

Due to the lexicographical ordering of the sistrings in the PAT array, the corresponding SPAT structure will also reflect this ordering. Thus, we need only to read and transfer to the main memory no more than 2 PAT array blocks, independently of the size of the answer. In the analysis that follows we consider only the answers bounded by a single block. The analysis for answers bounded by two blocks can be easily derived from the results that we present.

For the purpose of cost analysis we consider only the time spent in the secondary storage level. As discussed in the beginning of this Section, the cost in time units spent in the main memory level or higher, is entirely negligible. We also do not consider the transfer time between secondary storage device and main memory, which is also negligible when compared with the other two components of the access time: the seek time (t_s) and the rotational delay (t_r). This is true for both magnetic and CD-ROM disks.

The following definitions will be used in the cost analysis:

1. Let L_s indicate the length of one disk sector in bytes ($L_s = 2048$ for CD-ROM).
2. Let C_1 indicate the retrieval cost in t_s units when the PAT array is used as a single index level.
3. Let C_2 indicate the retrieval cost in t_s units when the two level hierarchy model is used.
4. Let G_2 indicate the feasible performance gain when the two level hierarchy model is used.

One disk sector is the minimum data block that can be transferred from disk to main memory in any case (magnetic or optical). Each PAT array entry is a *longint* 4 bytes pointer. Thus, each disk sector contains exactly $L_s/4$ PAT array entries. All cost computations will be done in terms of t_s units – seek time, so that any retrieval cost, given in seconds, will be divided by t_s. To find the answer we have to perform two binary searchs to find the exact left and right answer, each one taking $\log(b/2)$ disk accesses plus one additional access to the root of the tree. The cost expression C_2 for the two level model is:

$$C_2 = [2\log(b/2) + 1]t_s \qquad (2)$$

Expressing C_2 as t_s units we have:

$$C_2 = 2\log(b/2) + 1 = 2\log(b) - 1 \quad (3)$$

Substituting b by its definition given in Page 9, we have:

$$C_2 = f(n, M, l_s) = 2[\log(n) + \log(l_s) - \log(M)] - 1 \quad (4)$$

Considering a naive implementation for the PAT array with no additional index structure, and taking the disk sector size as the block size, we have:

Cost for index search: $[2\log(\frac{n/2}{L_s/4}) + 1]t_s$

Cost for text search: $[2\log(\frac{n/2}{L_s/4}) + 2\log(L_s/4) + 1]t_s$

So, the total cost in t_s units will be:

$$C_1 = f(n, L_s) = 2[2\log(n) - \log(L_s) + 1] \quad (5)$$

Comparing expressions (5) and (4), we can derive the improvement obtained from the two level hierarchy model, using the SPAT array structure. This performance improvement is given in terms of the number of times that the cost is reduced, which we call the performance gain factor (G_2):

$$G_2 = f(n, M, l_s, L_s) = \frac{C_1}{C_2} = \frac{2\log(n) - \log(L_s) + 1}{\log(n) + \log(l_s) - \log(M) - \frac{1}{2}} \quad (6)$$

Analyzing Equation (6) for a given text database with n index points and given values for M, l_s and L_s, the limit of G_2 is 2 when $n \to \infty$. This is an expected result because in the SPAT model we reduce the number of comparisons by a factor of at least 2. We can also see that G_2 is maximized if M is maximized or l_s is minimized, for a fixed value of the sector length, L_s. Table 3 shows the results for G_2 with n ranging from 1.6×10^6 to 102.4×10^6 index points (text files of about 10 to 600 megabytes), and $M = 4$ megabytes. For comparison purposes we consider $L_s = 2048$ bytes, which is the standard sector length for CD-ROM disks.

$M = 4$ megabytes, $r = 200 \times 10^3$, $l_s = 20$ bytes, $L_s = 2048$ bytes

Text size Megabytes	n Index points	$b = n/r$ Elements in PAT block	$B_p = b \times 4$ (bytes) PAT array block size	G_2
10	1.6×10^6	8	32	12.49
20	3.2×10^6	16	64	9.49
40	6.4×10^6	32	128	7.83
80	12.8×10^6	64	256	6.77
160	25.6×10^6	128	512	6.03
320	51.2×10^6	256	1024	5.50
640	102.4×10^6	512	2048	5.08

Table 3: Performance gain from the SPAT array (two-level hierarchy model)

5 Three Level Hierarchy Model

The three-level hierarchy model is an extension of the two-level model. The SPAT array index and the related PAT array are distributed among the main memory (first level), the magnetic disk (second level) and the CD-ROM disk (third level). This approach reduces to a minimum the height of the tree that has to be searched in the last phase in the optical disk, which is the slowest device in the memory hierarchy. Figure 5 illustrates the concept of the three-level hierarchy model. The primary index is extracted from the SPAT array and stored in main memory. This index is a small set of nodes of the SPAT array and its objective is to reduce the searching interval in the SPAT array file stored in the magnetic disk.

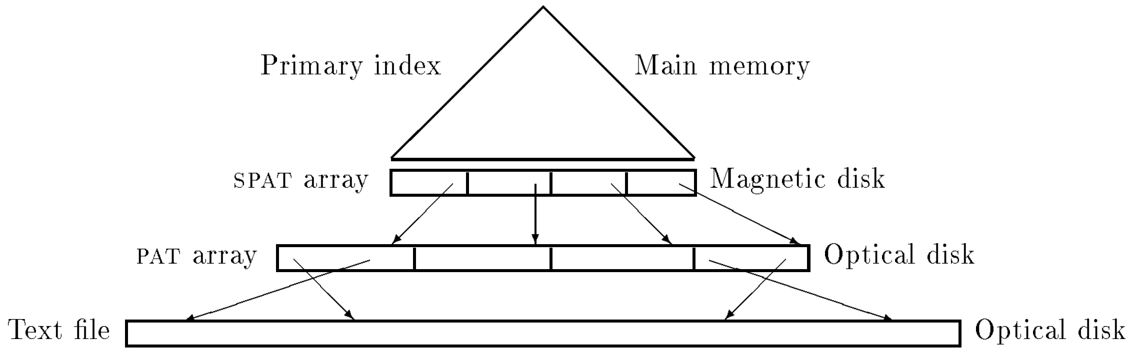

Figure 5: The concept of three-level hierarchy model for text retrieval in CD-ROM disk

The following parameters are needed to build up this index: (i) the maximum CD-ROM capacity; (ii) the maximum available space in the magnetic disk; (iii) the PAT array size (number of index points, n) and (iv) the entry length (l_s) in the SPAT array. ¿From a practical point of view we consider a 10 megabytes text file as a small database, a 100 megabytes text file as a medium database and a 300 megabytes text file as a large database. We also consider the beginning of every word of the text as an index point of interest and the average word size as containing 5 characters. Consequently, considering n as the number of PAT array pointers to words in the text, $n = 2 \times 10^6$ for small texts, $n = 20 \times 10^6$ for medium texts and $n = 60 \times 10^6$ for large texts. Considering these values, the heights of the binary search trees will range from 21 to 26 levels. As the last level is now a 600 megabytes CD-ROM disk, if we consider a 300 megabytes text file with a 60 million index points in the PAT array, then the total amount of information to be stored in the disk fills up to 540 megabytes. Thus, if no compression scheme is used for text or indices, text files should not be greater than 350 megabytes.

Ideally, we should try to maximize the number of levels of the search tree stored in main memory, so that most of the key comparisons would be done in the fastest device in the hierarchy, greatly reducing the retrieval time. However, we are faced with practical restrictions for the maximum search tree stored in the main memory, due to the limited amount of memory available and the time needed to transfer large files from CD-ROM disk to the main memory.

The primary index to be stored in main memory is constituted of a small set of selected SPAT array nodes. Notice that the entries in the primary index need to hold only the minimum string length that enables searching decisions in the binary tree. The minimum string length is the set of characters that the searching algorithm needs to discriminate the root of two adjacent subtrees at the next level in the hierarchy, which is stored in the magnetic disk. For large files, an adequate length for the main index entry seems to be approximately 30 characters.

For the two-level hierarchy model, a text file with $n = 5 \times 10^7$ index points ($h_{max} = 26$ levels) and a memory size of 2 megabytes available to store the SPAT array requires up to 18 CD-ROM accesses, in the worst case. This is the binary search that is needed to find the exact left and right bounds in a PAT array block with $b = 512$ elements, or $2 \log b = 18$. Using the three-level hierarchy structure we can reduce this last search phase in the CD-ROM to 6 disk accesses for medium text files and to 8 disk accesses for large text files. If we select 6 levels for the last searching phase to be done in the CD-ROM we can distribute the remaining levels of the tree between magnetic disk and main memory. As an example, the following values may be used: $h_1 \leq 9$ levels in the main memory (primary index), $h_2 \leq 11$ levels in the magnetic disk (SPAT array + primary index), and $h_3 \leq 6$ levels in the optical disk (text + PAT array + SPAT array + primary index).

The searching algorithm for the three-level hierarchy model is:

```
1.  Read the SPAT array file from optical disk to magnetic disk;
2.  Read the primary index file to main memory;
3.  Repeat
   3.1.  Perform binary search in the primary index; {Cost ≈ 0}
   3.2.  Perform binary search in the magnetic disk and find left and right
         bounds in the SPAT array;
   3.3.  Transfer left and right blocks of the PAT array from optical disk
         to main memory;
   3.4.  With the left block do binary search between main memory and the
         text file; {find exact left bound}
   3.5.  With the right block do binary search between main memory and the
         text file; {find exact right bound}
   3.6.  Read the occurrences from the text file as desired;
   Until {end of queries}
```

Ideally, all the PAT array search tree should be transferred to the magnetic disk, which is much faster. But this is not a practical solution due to the following reasons: (i) A PAT array for a large text file (e.g. 300 megabytes) would take too long to be transferred from optical do magnetic disk. At the maximum rate of 300 kilobytes per second, this would take more than 15 minutes; (ii) Such a large file would require too much space in the magnetic disk and (iii) The text remains in the optical disk, so that a lot of slow seeks would have still to be performed. Thus, it is of practical interest to keep only the SPAT array in the second level – the magnetic disk –, which is not greater than 10% of the corresponding PAT array file.

¿From the previous discussion we can distinguish the following advantages of the three-level hierarchy model: (i) The demand for main memory is reduced because a larger part of the search tree is now stored in the magnetic disk; (ii) The SPAT array can be larger due to the less expensive and larger capacity magnetic disk and consequently, the SPAT array index is a much more representative version of the text and the PAT array file, and (iii) Frequently used text database in optical disks could have their SPAT array index and primary index permanently stored in the magnetic disk, saving the time to load these files each time a CD-ROM is used. Considering that the size of the SPAT array file is not greater than 10% of the corresponding PAT array, a 600 megabytes magnetic disk can store indices corresponding to a 6 gigabytes PAT array file.

Analytical Results

In this Section we present a general cost expression for the three-level hierarchy model. The complete index structure, represented by the PAT array, can be viewed as a distributed tree, where part of it is stored in the main memory (first level), another part is stored in the magnetic disk (second level), and the rest of it is stored in the optical disk (third level). To compare the performance of this model with the two-level hierarchy model, we have to consider the part of the SPAT array stored in the magnetic disk, and compute this additional cost. The worst case expression can be derived from the sum of the cost to traverse all the tree from the root to the leaves, considering the specific costs to traverse each level in the memory hierarchy. We neglect the computation that is done in the main memory, since access time for the second and third levels in the hierarchy are from 5 to 6 orders of magnitude greater.

The following definitions will be used in the cost analysis:

1. C_3 = retrieval cost in t_s units when the three level hierarchy model is used.
2. G_3 = feasible performance gain when the three level hierarchy model is used.
3. $p = t_{s(mag)}/t_{s(opt)}$ = seek cost ratio between magnetic and optical disk;
4. h_2 = height of the search tree in the magnetic disk (second level);
5. h_3 = height of the search tree in the optical disk (third level);
6. D = magnetic disk space available for the SPAT array (second level);
7. $r_D = D_{size}/l_s$ = number of elements in the SPAT array (magnetic disk);
8. $r_M = M/l_s$ = number of elements of the SPAT array in the main memory;
9. $B = n/r_D = (nl_s)/D_{size}$ = number of elements in a PAT array block.

We only consider the case in which the answer for a query is bounded by one PAT array block, and each block is not larger than one CD-ROM sector. This last assumption is quite reasonable, since the introduction of the second level in the magnetic disk (virtually the same as increasing the main memory available, M, in the two-level hierarchy model) increases the space to store the index, which, in turn, reduces the size of the PAT array blocks.

¿From the above definitions, we derive the expressions for the height of the binary search tree in the second and third levels (h_2 and h_3):

$$h_2 = \log(r_D) - \log(r_M) \tag{7}$$

$$h_3 = \log(B) \tag{8}$$

As we are neglecting the costs for the first level, the overall cost to traverse the tree and find the left and right bounds of an occurrence is twice the cost needed to traverse the second and third level from the root to the leaves. This cost, in $t_{s(opt)}$ units, is given by:

$$C_3 = 2(h_2 \times t_{s(mag)} + h_3 \times t_{s(opt)}) = 2(ph_2 + h_3)t_{s(opt)} \quad (9)$$

Substituting h_2 and h_3 in Equation 9 and using the definitions of r_D, r_M and B, we obtain:

$$C_3 = f(n, l_s, M, D, p) = 2[\log(n) + \log(l_s) + (p-1)\log(D) - p\log(M)] \quad (10)$$

We can now compare the cost C_3 with the retrieval cost using the PAT array as a single level index, C_1, derived in Section 4. Neglecting the transfer time for a single block (one CD-ROM sector), we have:

$$G_3 = f(n, l_s, L_s, M, D) = \frac{C_1}{C_3} = \frac{2\log(n) - \log(L_s) + 1}{\log(n) + \log(l_s) + (p-1)\log(D) - p\log(M)} \quad (11)$$

The following example illustrates the improvement obtained with this model: Let $n = 50 \times 10^6$ index points, $M = 2$ megabytes, $D = 16$ megabytes, $l_s = 30$ characters and $p = 0.03$. Using Equations 10 and 4 we obtain: $C_2 = 18t_{s(opt)}$ and $C_3 = 13t_{s(opt)}$. Thus, the retrieval cost in the worst case is reduced by 1.4 times when compared with the two-level hierarchy model. If we compare this result with the naive single level implementation of the PAT array in the CD-ROM, the overall improvement increases to 6.2 times (or 84% improvement).

Table 4 shows the results for various text file sizes and compares the performance gain achieved using the two-level hierarchy model and the three-level hierarchy model. All the calculations were made considering the worst case and having the answer bounded by one PAT array block. Even though the magnetic medium is a much slower device as compared to the main memory, performance improvement has been achieved with the three-level approach mainly due to two factors: (i) The space available to store the index in the magnetic disk is much larger, so that the SPAT array may exhibit a finer index granularity, which signifies that the index is a much more representative version of the PAT array and the text database, and (ii) The magnetic disk is 15 to 30 times faster than the optical device.

$M = 1$ megabyte, $D = 16$ megabytes, $l_s = 30$ bytes, $p = 0.03$

Text size (Megabytes)	Index points (n)	C_1 (t_s)	C_2 (t_s)	C_3 (t_s)	Gain (G_3) C_1/C_3
10	2×10^6	63	10	4	15.8
20	4×10^6	67	12	6	11.2
40	8×10^6	71	14	8	8.9
80	16×10^6	75	16	10	7.5
150	30×10^6	79	18	12	6.5
300	60×10^6	83	20	14	5.9

Table 4: Performance gain using the two and three-level hierarchy model

6 Conclusions and Future Work

In this paper we presented two indices hierarchies and the corresponding searching algorithms to improve retrieval from full text databases stored in secondary devices such as magnetic or optical disks. For large text files stored in CD-ROM optical disks and considering the poor retrieval performance of such disks, we conclude that the use of hierarchical models results in significant advantages to the end user. The cost analysis carried out for the two-level hierarchy model and for the three-level hierarchy model were compared to the cost of the naive single level PAT array implementation for CD-ROM devices. The two-level hierarchy model is particularly interesting for small to medium size text databases, since the amount of memory required to achieve good improvement in response time is proportional to the size of the associated PAT array. The three-level hierarchy model is noticeably interesting for large text databases and full text retrieval systems that accesses a frequently used set of CD-ROM disks. In such a situation, the second level of the index hierarchy could be permanently stored in a magnetic disk, saving the time needed to transfer it from the CD-ROM to the magnetic disk. Considering that the SPAT array requires only a small fraction of the storage space for the corresponding PAT array, we estimate that a single magnetic disk can hold enough indices files to access a large number of CD-ROM disks.

There are two improvements that can be implemented to reduce space and time complexities of the model that we have just described. The first one is a compression of the SPAT array index that reduces the demand for space in the main memory. It has been shown that the entries of the higher level indices are formed by the first l_s characters of the sistring pointed by the last element of each PAT array block. In order to perform the correct binary search we have to guarantee that no consecutives entries in the SPAT array are equals. We have seen from the analytical results that the smaller is the size of a PAT array block the higher is the improvement in retrieval performance. However, by reducing b, we bring each SPAT array entry nearer to adjacent ones, so that the first l_s character of the corresponding sistring in the text may not be sufficient to make two consecutive entries distinct, as it is necessary. One immediate solution is to increase l_s up to the length in which we do not have two equal SPAT array entries. As a consequence, the demand for memory becomes still greater, and the space complexity might be critical. A better approach is to make use of the existence of the common prefixes in consecutives SPAT array entries, and store only the differential part of them. In this case the length of the SPAT array entries will be variable. Consider, for instance, the following consecutive SPAT array entries, with $l_s = 26$ chars:

```
1 - THIS_IS_A_SPAT_ARRAY_INDEX
2 - THIS_IS_A_SPAT_INDEX_ENTRY
3 - THIS_IS_A_SPAT_INDEXING_PO
```

The prefix *THIS_IS_A_SPAT_* is the same for the 1^{st} and 2^{nd} entries, and the prefix *THIS_IS_A_SPAT_INDEX_* is the same for the 2^{nd} and 3^{rd} entries. Thus, if we selet the first entry to be stored in the SPAT array with all its 26 characters, then the second entry needs to store only the position where the first differential symbol occurs and the symbol itself. In this example, as the difference occurs in the position 16 of the string, the second entry would contain the information *16-I*, which is easily processed for comparison purposes during the search in the SPAT array. The same idea is repeatedly applied for subsequent entries, bearing in mind that we may need to keep track of the changes of the previous entries, which may be necessary to store more than one character in the differential part. In this example the third entry needs to store the sequence *INDEX_* Hfrom positions 16 to 21 to make possible to reconstruct the third entry, which differs from the second in position 21. We have also to select an adequate number of full SPAT array entries to be stored, to make easier the reconstruction of any other differential entries in the index.

The other improvement, related to the reduction of the time complexity, may be achieved when the last level of the hierarchy is the CD-ROM disk. The binary search in the optical disk can be more efficient if we consider the cost function of this device, which is strongly dependent of disk position in the CD-ROM, as we described in Section 2. In both two and three level hierarchy models, the first phase of the searching process is performed in the faster devices: the main memory or the magnetic disk, respectively. The answer found at these first levels produces the set of all the positions (b PAT array pointers) that we have to visit on the disk to find the exact bounds of the desired answer. By considering the set of positions to visit in the CD-ROM, and the seek cost function of the device, it is possible to apply the principle of the weighted binary tree to devise an algorithm that produces an optimal searching schedulling in the disk tracks. As a result, the total distance traveled by the optical head is reduced and the retrieval performance is better. We are actually working on these ideas and other topics related to the index and text compression and the optimization of the binary search in the CD-ROM disk.

Acknowledgements

The authors wish to acknowledge the financial support from the Brazilian CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, IBM do Brasil, and Programa de Cooperación Científica Chile-Brasil de Fundación Andes.

References

Ars Innovandi (1992) *Search City 1.1 - Text Retrieval For Windows™ Powers Users*, Reference Guide, Santiago, Chile.

Baeza-Yates, R., Barbosa, E. F., Ziviani, N. (1993) "Hierarchies of Indexes for Text Searching in Read-Only Optical Disks" Proceedings of the *First South American Workshop on String Processing*, Belo Horizonte, Brasil, September 1993, 25–41.

Barbosa, E. F., e Ziviani, N. (1992) "Data Structures and Access Methods for Read-Only Optical Disks", in *Computer Science: Research and Applications*, R. Baeza-Yates and U. Manber (eds), Plenum Publishing Corp., 189–207.

Gonnet, G. H. (1983) "Unstructured Data Base for Very Efficient Text Searching", *Proceedings of the 2nd ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, Atlanta, Georgia, 117–124.

Gonnet, G. H. (1987) "PAT 3.1: An Efficient Text Searching System", User's Manual, Center for the New OED, University of Waterloo.

Gonnet, G. H. and Baeza-Yates, R. (1991) *Handbook of Algorithms and Data Structures*, Addison-Wesley.

Gonnet, G. H., Baeza-Yates, R. and Snider, T. (1992) "PAT Arrays: An Alternative to Inverted Files", in *Information Retrieval*, Frakes and R. Baeza-Yates (Eds), chapter 5, Prentice-Hall.

Katz, R. H., Gibson, G. A. and Patterson, D. A. (1989) "Disk System Architeture for High Performance Computing", *Proceedings of the IEEE*, Vol. 77, No. 12, December 1989, 1842–1858.

Manber, U. and Myers, G. (1990) "Suffix Arrays: A New Method for On-Line String Searches", *ACM-SIAM Symposium on Discrete Algorithms*, January 1990, 319–327.

Morrison, D. R. (1968) "PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric", *Journal of the ACM 15(4)*, 514–534.

Perleberg, C. H. (1993), "Implementation of Optical Disk PAT Arrays", *Unpublished tecnical report*, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil.