Pergamon

Information Systems Vol. 21, No. 6, pp. 497–514, 1996

0306-4379/96 $15.00 + 0.00

HIERARCHIES OF INDICES FOR TEXT SEARCHING†

RICARDO BAEZA-YATES[1], EDUARDO F. BARBOSA[2] and NIVIO ZIVIANI[2]

[1]Departamento de Ciencias de la Computación Universidad de Chile, Santiago, Chile
[2]Departamento de Ciência da Computação Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

(Received 17 August 1994; in final revised form 2 August 1996)

Abstract — We present an efficient implementation of a recently known index for text databases, 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 (or suffix array). Considering that text searching in a large database spends most of the time accessing external storage devices, we propose additional index 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 main memory and one level of external storage (magnetic or optical devices) and a three-level hierarchy model that uses 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 and experimental results are presented for both models. For 160 megabytes of text stored on CD-ROM disk the two-level model using 2 megabytes of main memory costs 20% of the PAT array used as a single level.Copyright ©1996 Elsevier Science Ltd

Key words: Index Hierarchy, Memory Hierarchy, Text Searching, PAT Arrays, Suffix Arrays, Magnetic Disks, Read-Only Optical Disks, CD-ROM

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 databases one of the fastest growing category of databases. For static databases it is worthwhile to preprocess the database and build an index to decrease search time. A new and simple type of index is the PAT array [9, 10] or suffix array [15]. 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 [17] built on all suffixes of a text database. The PAT tree, also called suffix tree, was originally described by [14].

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 for a word in a PAT array requires at most $4 \log n$ disk accesses. The extra space used by a PAT array is only one pointer per indexing point, which is approximately 60% of the space occupied by a large text database (less if stop words are used).

The two most important text searching methods are signature indices (or files) and lexicographical indices (for example, inverted files and PAT arrays). Signature indices use hashing techniques to produce an index with small space overhead, between 10% and 20% of the text size. However, there is one problem: some answers may not match the query, a problem known as false drop [6].

A two-level searching method, called Glimpse, was proposed in [16]. Glimpse combines a partial inverted file with sequential searching. The text is divided in blocks of the same size and a table of all different words and a list of addresses indicating the blocks where each word appears is built. The idea is a hybrid between full inverted files and pure sequential search with no indexing. To search for a word it is necessary to look first in the inverted file and then use sequential search in all the corresponding blocks. In the worst case, it may be necessary to search all the blocks, which makes Glimpse inadequate to be used for very large texts (say over 250 megabytes).

†Recommended by Stavros Christodoulakis

There are several structures that can be used in implementing lexicographical inverted files, such as sorted arrays, prefix B-trees and tries [13]. Compared with sorted arrays, prefix B-trees and tries use more space. On the other hand, updates are easier in B-trees than in sorted arrays, and cannot be done efficiently in tries and PAT arrays. Tries are very efficient because the search is directed by the query itself, giving a search time proportional to its length, instead of logarithmic in the number of words. The main problem with tries is that they are very time consuming to build and use more space, making them impractical for large texts.

In general inverted files need a storage overhead between 30% and 100%, depending on the data structure and the use of stop words, and the search time is logarithmic. Similar space and time complexity can be achieved by PAT arrays. The great advantage of PAT arrays is its potential use in other kind of searches that are difficult or inefficient over inverted files. That is the case when searching for a long sequence of words, some types of boolean queries, regular expression searching, longest repetitions and most frequent searching [11]. Consequently, PAT arrays should be considered seriously when designing a searching method for text databases that are not updated frequently.

The main objective in the design of any memory system is to obtain adequate storage capacity, with a 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 construct 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 summarized as follows: larger capacity devices are associated with low cost and slow access time memories, while smaller capacity devices are associated with faster access and higher cost memories. We have to emphasize that the current bottleneck of large databases is I/O speed, because the processing speed has increased much faster.

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, low cost to the end user and good physical and logical integrity. However, seeking in the CD-ROM is a very 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 poor performance from a practical point of view.

The goal 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 index 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 are stored on faster memories and only a few index levels are left for the slowest device in the hierarchy. This approach effectively reduces the time required and the number of disk accesses needed to answer a query, thus improving the searching performance. The hierarchy design was presented in [1, 2].

In Section 2 we briefly describe the diversity of external storage device performances with emphasis on 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 disks as the second level. In Section 5 we present the three-level hierarchy model, which distributes the index 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. In Section 6 we present experimental results for both models.

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 a good level of performance at the lowest possible cost. 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. External storage devices, such as magnetic and optical disks, normally constitute the last level in the memory hierarchy, where the database of interest is normally stored.

Although the costs vary between magnetic and optical disks, in both cases they have three components: *seek time*, *latency time*, and *transfer time*. Seek time is the time needed to move the disk head to the desired disk track and therefore depends on the current head position. Latency time (or rotation time) is the time needed to wait for the desired sector to pass under the disk head. The average latency time is constant for magnetic disks and variable for CD-ROMs, which rotate faster reading inner tracks than when reading outer tracks. In our analysis, we use an average CD-ROM latency. Transfer time is the time needed to transfer the desired data from the disk head to main memory.

The access time, t_a, needed to read n_s sectors from track c, with the reading mechanism being currently on track b is

$$t_a = seek(|b - c|) + latency + n_s \times transfer$$

For analytical purposes, we selected three levels in the memory hierarchy to build our model: main memory, magnetic disk, and CD-ROM disk. This choice was based on the following: first, the difference in access time from one level to the next should be at least one order of magnitude. Because main memory is more than 200,000 times faster than the faster secondary storage device (the magnetic disk) and more than 4 million 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.

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. Second, we do not consider the write-once read-many (WORM) disk because it is not of practical use for large textual database 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. 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 ± 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 160 to 450 milliseconds due to optical head displacement, against approximately 8 to 15 milliseconds for a typical magnetic disk. More details on the differences between magnetic disks and CD-ROM disks can be found in [3, 7].

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 [3]. If we consider that more complex queries may require a very large number of disk accesses, text retrieval in large databases stored on magnetic disks will also present better performance if we reduce the number of disk accesses neccesary to answer the query.

3. PAT ARRAYS OR SUFFIX ARRAYS

The traditional text model 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 full-text databases sees the text as one long string [14, 15, 10]. Each position in the text is called a *semi-infinite string* or *suffix*. A suffix 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. Suffixes are compared lexicographically character by character, and so any two strings in different positions do not compare equally.

Depending on the character sequences that are likely to be suffixes 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 suffix.

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

Fig. 1: A simple text database with nine index points

A PAT array is an array of pointers to suffixes, providing sorted accesses to all suffixes of interest in the text. With a PAT array it is possible to obtain all the occurrences of a string prefix (of a suffix) in a text in logarithmic time using binary search. The binary search is indirect since it is necessary to access both the index and the text to compare a search key with a suffix. Figure 2 shows the PAT array for the sample text database presented in Figure 1.

1	2	3	4	5	6	7	8	9
28	14	38	17	11	25	6	30	1

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

Fig. 2: PAT array or suffix array

To search for the prefix *tex* in the PAT array presented in Figure 2 we perform an indirect binary search over the array and obtain [7, 8] as answer (position 7 of the PAT array points to the suffix beginning at the position 6 in the text, and position 8 of the PAT array points to the suffix 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. Searching for a string in a PAT array takes at most $2\,\ell_q \log n^{\dagger}$ character comparisons and at most $4 \log n$ disk accesses, where n is the number of indexing points and ℓ_q is the length (number of characters) of the string query. 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. Building a PAT array is similar to sorting variable length records, at a cost of $O(n \log n)$ accesses on average. The extra space used by a PAT array is only one pointer per index point.

---

†All logarithm functions used in this paper are base 2.

4. TWO-LEVEL HIERARCHY MODEL

In this section we consider a two-level memory hierarchy model composed of main memory and a secondary storage device – either magnetic or optical disk.

4.1. Description of the Model

Our solution to improve the performance of the searching process is to divide the PAT array into equal-sized blocks and move one element of each block to main memory, together with additional information about the text related to the element selected from each block. 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 performed in main memory. As a result, most of the searching work is done in main memory, thus reducing the number of disk accesses.

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 the suffix pointed to by the last entry of a SPAT array block.

We now define certain measures:

1. Let n indicate the number of index points in the text.
2. Let M indicate the main memory available to store the SPAT array index, in bytes.
3. Let ℓ_s indicate the number of characters of a SPAT array entry.
4. Let ℓ_q indicate the number of characters of a given query.
5. Let L_s indicate the length of one disk sector in bytes ($L_s = 2048$ for CD-ROM).
6. Let r indicate the number of elements of the SPAT array ($r = M/\ell_s$).
7. Let b indicate the number of elements in a PAT array block ($b = n/r = (n \times \ell_s)/M$).
8. Let b_R indicate the number of repeated SPAT array entries for a given query.

Figure 3 illustrates a SPAT array index for the PAT array of Figure 2. We divide the PAT array in 3 blocks, each one with 3 entries. For each PAT array block we transfer ℓ_s characters from the suffix pointed to by the last PAT array entry in this block ($\ell_s = 3$ in this example) to the SPAT array. 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 presents the algorithm to build the SPAT array structure from an existing PAT array index. The size of the main memory (M) to be used by the SPAT array structure is an important parameter. In our experiments we used sizes for M in the range of 1 to 8 megabytes. If we consider memory overhead as the main memory space M divided by the PAT array space ($n \times 4$ bytes) then a 2 megabytes memory space to accommodate the SPAT array for a text with $n = 50$ million indexing points gives an overhead of 1%. In this case, the time to transfer the SPAT array from a magnetic disk with a transfer rate of 2 megabytes/second is 1 second, and 3.6 seconds from a CD-ROM disk with a transfer rate of 600 kilobytes/second.

The choice of the size of ℓ_s is another important parameter. From the above definitions, for a text with n index points, a SPAT array with r entries and an available memory M, the value for ℓ_s is a linear function of b, as follows: $\ell_s = (M/n)b$. From the performance point of view, the smaller is the PAT array block b the higher is the performance gain, since the most costly searching is done within the elements of this block and the text stored on disk. However, when the size of b decreases, the number of SPAT array entries increases proportionally.

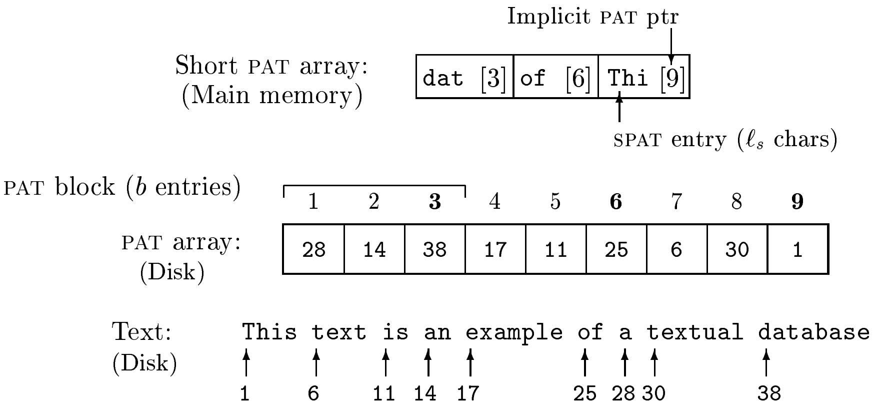

Fig. 3: Short PAT array (SPAT)

Define main memory size (M) to be used by the SPAT array;
Select number of characters for each SPAT array entry (ℓ_s);
Compute the number of elements of the SPAT array $(r = M/\ell_s)$;
For $i = 1$ to r do
    For each i block of the PAT array
        Copy the first ℓ_s chars of the suffix corresponding to the
        $(b \times i)^{th}$ entry of the PAT array to the i^{th} entry of the SPAT array;

Fig. 4: Algorithm to build the SPAT array

On the other hand, text characteristics significantly affects the adequate choice for ℓ_s. Intrinsic characteristics of the text (text size, contents, language, style, etc.) may require a large value for b in order to differentiate consecutive SPAT entries, because some parts of the text may have too many suffixes that are equals up to a large number of characters. As a consequence, the minimum value for ℓ_s such that no two consecutive PAT array entries are equals might be large, thus increasing the demand for main memory. Given the available memory size M and the text, it is difficult to find analytically the best value for ℓ_s to meet the contradictory requirements involved since the text cannot be modeled as a true random combination of the symbols of the alphabet.

In the following we discuss ways to obtain an adequate value for ℓ_s. The first information we need is the distribution of the number of repeated suffix entries for different values of ℓ_s. This can be obtained during the construction of the PAT array, or using the following algorithm: we read the PAT array from 1 to n step by b_{min}. The value for b_{min} is obtained by using ℓ_s as the average word size in the text in the previous definition of b. For every step we compare the two suffixes to find out the number of characters that are equals. The cost is $O(n/b_{min})$ disk accesses. A 600 megabytes text file with $n = 10^8$ index points, $M = 1$ megabyte and $b_{min} = 60$ entries, takes approximately 28 minutes to perform the text scanning, considering 10 ms per random disk access.

The largest number of repeated text suffixes b_R of length ℓ_s is obtained for $6 \leq \ell_s \leq \ell_{s(PAT)}$, where 6 is the average word size and $\ell_{s(PAT)}$ is the number of characters used for suffix comparison to build the PAT array. Next, we obtain the function $\ell_s = f(b)$ for values of b so that $0 \leq b_R \leq k$, where k is an user defined parameter. Figure 5 illustrates this approach for a 12 megabytes text file. The straigth dashed line represents all the solutions for ℓ_s and b that satisfies the SPAT parameters. The solid curve represents the function $\ell_s = f(b)$ for actual text when $b_R = 0$ (no repeated entries occur in the index), and the dashed curve represents the same function, but allowing up to k repeated entries in the index. The intersection points (ℓ_{s1} and ℓ_{s2}) are two solutions that satisfy both the parameters of the model and the actual text characteristics. Table 1 presents practical values for ℓ_s for different text sizes. The additional cost for allowing repeated entries in the SPAT array is compensated by smaller block sizes, thus requiring fewer disk accesses. The values presented in Table 1 were confirmed analytically in [18].

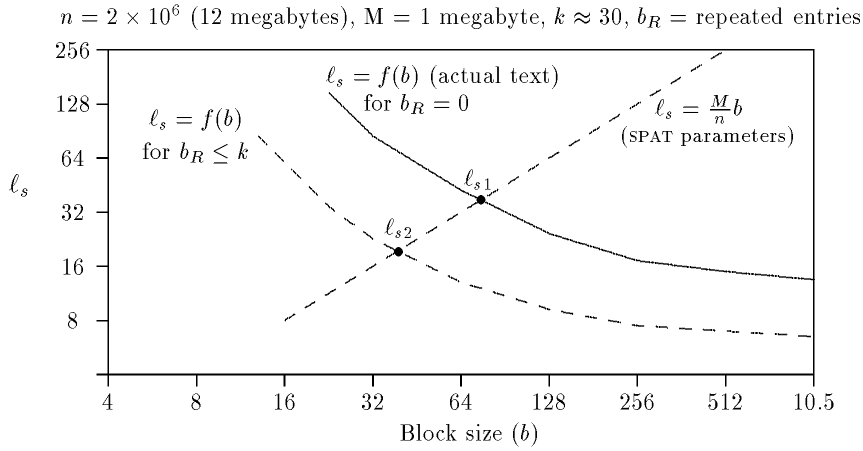

Fig. 5: Estimation of ℓ_s from text characteristics and SPAT parameters

Text size (MB)	Repeated entries (b_R)	SPAT entry (ℓ_s)	Block size (b)	Memory (KB)
8	0	15	1024	20
	≤ 15	10	128	104
16	0	20	1024	52
	≤ 16	20	64	833
32	0	30	1024	156
	≤ 20	15	64	1250
64	0	30	1024	313
	≤ 16	15	128	1250

Table 1: Practical values for ℓ_s

Figure 6 presents the algorithm to search using the SPAT array structure. The search is done in two phases: first, the SPAT array is transferred to the main memory, where a binary search is performed in the reduced index structure, with no disk accesses. If there are repeated entries in the SPAT array we have to consider the special case where $\ell_q > \ell_s$ (the length of the search key is larger than ℓ_s). In this case, we read from disk to main memory the last pointer of all PAT array blocks pointed to by the repeated SPAT array entries. This requires a single disk access, because the PAT array is stored contiguously on the disk. Next, we perform a binary search on the text suffixes pointed to by the PAT array pointers of repeated SPAT array entries and locate the exact bounds, at the cost of $\log b_R$ disk accesses. Figure 7 illustrates this case, where $b_r = 5$ and $\ell_s = 4$. The query has 6 characters and the 5 repeated SPAT array entries match the first four characters of the query. This search finds the SPAT array entry that points to a small PAT array block that contains the desired answer.

Second, the PAT array block encountered in the first search phase is moved from disk to main memory. Finally, a second binary search is performed between memory (PAT array block containing the answer) and the disk (text file), where the exact bounds of the answer are found. If the answer has more than b occurrences it may be bounded by at most two SPAT array entries. In this case, left and right blocks are used in the last phase of the binary search, following the same procedure.

```
Read SPAT array file to main memory; {Done only once}
Repeat
  Find left and right bounds in the SPAT array; {Cost ≈ 0}
  If (ℓq > ℓs) and (bR > 1) then
   Search for left and right blocks among the bR text suffixes; {Cost = log bR}
  Transfer left and right blocks of PAT array from disk to memory;
  With left block do binary search between memory and text file
        {find exact left bound}
  With right block do binary search between memory and text file
        {find exact right bound}
  Read the occurrences from text file as desired;
Until {End of queries}
```

Fig. 6: Searching algorithm for the SPAT array

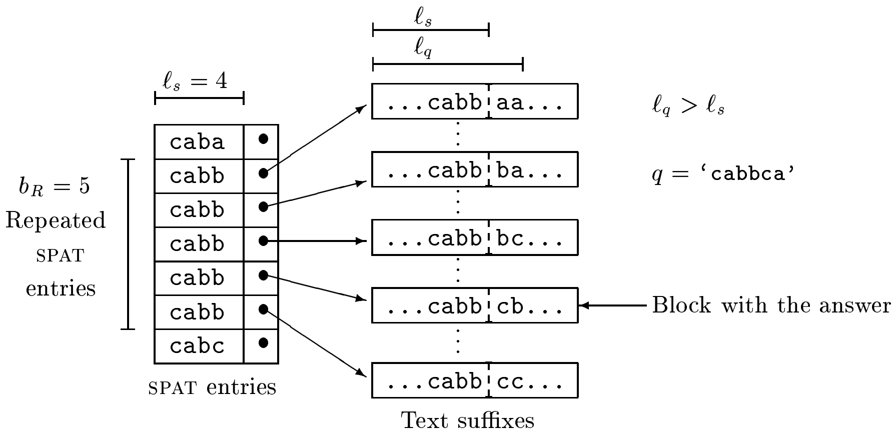

Fig. 7: Searching scheme with repeated entries in the SPAT ($\ell_q > \ell_s$)

4.2. Analytical Results

Due to the lexicographical ordering of the suffixes in the PAT array, the corresponding SPAT structure will also reflect this ordering. Thus, we need only read and transfer to main memory no more than 2 PAT array blocks, independent of the size of the answer. These two blocks correspond (in the worst case) to the bounds of the desired answer in the the SPAT array. When searching for a small prefix in a large text, we have a high probability of finding many disperse occurrences in the text and the answer might be bounded by two PAT array blocks. In this case, the exact left and right bounds are found with $\log b$ disk accesses in each block, plus one additional access in each block to start the binary search. Conversely, when we are searching for a large key, perhaps containing many words, the probability of finding many occurrences is very small and the answer might be in a single PAT array block. In this case, the exact left and right bounds are found with $\log(b/2)$ disk accesses in each half of the block plus one additional access to start the binary search. In the analysis that follows we consider only the answers bounded by a single block.

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 devices 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 C_1 be the retrieval cost in t_s units when the PAT array is used as a single index level.
2. Let C_2 be the retrieval cost in t_s units when the two level hierarchy model is used.
3. Let $G_2 = \frac{C_1}{C_2}$ be the feasible performance gain compared to the standard PAT array 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 4 byte pointer. Let L_s be the size of a disk sector. 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 searches 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:

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

Expressing C_2 as t_s units we have:

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

Substituting b by its definition we have:

$$C_2 = f(n, M, \ell_s) = 2[\log n + \log \ell_s - \log M] - 1 \tag{1}$$

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 is the sum of the cost for index search plus the cost for text search, as follows:

$$C_1 = f(n, L_s) = 2[2\log n - \log L_s + 1] \tag{2}$$

Comparing Eq. (2) and Eq. (1) we can derive the improvement obtained from the two-level hierarchy model, as follows:

$$G_2 = f(n, M, \ell_s, L_s) = \frac{C_1}{C_2} = \frac{2\log n - \log L_s + 1}{\log n + \log \ell_s - \log M - \frac{1}{2}} \tag{3}$$

Text database on CD-ROM

$M = 2$ megabytes, $r = 200 \times 10^3$, $\ell_s = 20$ bytes, $L_s = 2048$ bytes

Text size megabytes	Index points n	Gain G_2 C_1/C_2
10	1.6×10^6	8.9
20	3.2×10^6	7.4
40	6.4×10^6	6.4
80	12.8×10^6	5.7
160	25.6×10^6	5.2
320	51.2×10^6	4.9

Table 2: Performance gain from the two-level hierarchy model

Analyzing Eq. (3) 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 2 shows the results for G_2 with n ranging from 1.6×10^6 to 51.2×10^6 index points (text files of about 10 to 600 megabytes), and $M = 2$ megabytes. For comparison purposes we consider $L_s = 2048$ bytes, which is the standard sector length for CD-ROM disks.

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 main memory (first level), magnetic disk (second level) and CD-ROM disk (third level). As the last level is now a 600 megabytes CD-ROM disk, if we consider a 300 megabytes text file with 60 million index points in the PAT array, then the total amount of information to be stored on the disk occupies 540 megabytes. Thus, if no compression scheme is used for text or index, text files should not be greater than 350 megabytes.

5.1. Description of the Model

The three-level hierarchy approach minimizes the height of the tree that must be searched in the last phase in the optical disk, which is the slowest device in the memory hierarchy. Figure 8 illustrates the concept of the three-level hierarchy model. The primary index is extracted from the SPAT array and stored on main memory. Its objective is to reduce the searching interval in the SPAT array file stored on the magnetic disk.

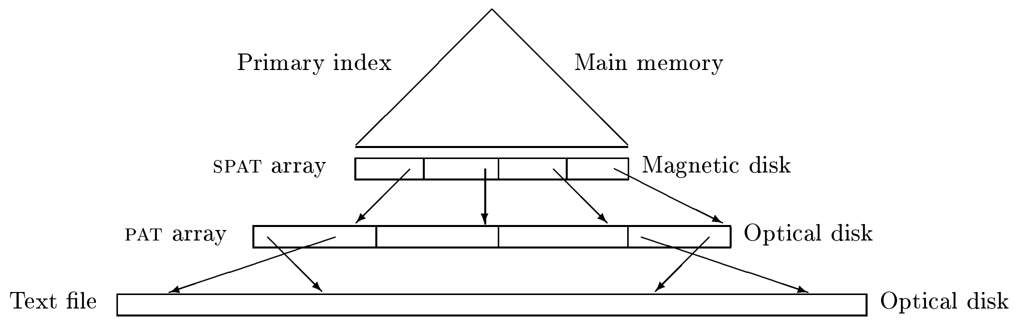

Fig. 8: Three-level hierarchy model for text retrieval in CD-ROM disk

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. Ideally, we should try to maximize the number of levels of the search tree stored on main memory, so that most of the key comparisons would be performed 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 on 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 on main memory consists of a small set of selected SPAT array nodes. Notice that the entries in the primary index need 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 on the magnetic disk. For large files, the adequate length for the main index entry depends on the size of the text, its nature (idiom, style, contents, etc.) and the PAT array block size. Notice that the PAT array block size is a measure of the distance between two consecutive SPAT entries. In practice, for a given text of fixed size, we observe that the larger this distance is, the smaller is the minimum length of the main index entry.

For the two-level hierarchy model, a text file with $n = 5 \times 10^7$ index points ($\log n = 26$ binary search levels) and a memory size of 2 megabytes 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. Let h_i, $1 \leq i \leq 3$, be the number of binary search levels in main memory, magnetic disk and optical disk, respectively. 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 algorithm to build the SPAT array structure for the three-level hierarchy model is similar to the algorithm for the two-level hierarchy model presented in the previous section.

Figure 9 presents the searching algorithm for the three-level hierarchy model.

```
Read SPAT array file from optical to magnetic disk {Done only once};
Read primary index file to main memory {Done only once};
Repeat
  Perform binary search in the primary index; {Cost ≈ 0}
  Perform binary search in the magnetic disk and find left and
         right bounds in the SPAT array;
  If (ℓ_q > ℓ_s) and (b_R > 1) then
  Search for left and right blocks among the b_R text suffixes; {Cost = log b_R}
  Transfer left and right blocks of the PAT array from optical
         disk to main memory;
  With left block do binary search between main memory and
         the text file; {find exact left bound}
  With right block do binary search between main memory and
         the text file; {find exact right bound}
  Read the occurrences from the text file as desired;
Until {end of queries}
```

Fig. 9: Searching algorithm for the three-level model

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 on 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; (iii) Frequently used text databases on optical disks could have their SPAT array index and primary index permanently stored on magnetic disk, saving the time to load these files each time a CD-ROM is used.

5.2. Analytical Results

Now 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 on the main memory (first level), another part is stored on the magnetic disk (second level), and the rest of it is stored on 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 on the magnetic disk, and compute this additional cost. The worst case expression can be derived from the sum of the cost to traverse the entire 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 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. Let C_3 be the retrieval cost in t_s units when the three-level hierarchy model is used.
2. Let $G_3 = \frac{C_1}{C_3}$ be the feasible performance gain compared to the standard PAT array when the three-level hierarchy model is used.
3. Let $p = t_{s(mag)}/t_{s(opt)}$ be the seek cost ratio between magnetic and optical disk, where $t_{s(mag)}$ and $t_{s(opt)}$ represent the seek times for magnetic disk and optical disk, respectively.
4. Let h_2 be the height of the search tree in the magnetic disk (second level).
5. Let h_3 be the height of the search tree in the optical disk (third level).
6. Let D be the magnetic disk space available for the SPAT array (second level).
7. Let $r_D = D/\ell_s$ be the number of elements in the SPAT array (magnetic disk).
8. Let $r_M = M/\ell_s$ be the number of elements of the SPAT array in the main memory.
9. Let $B = n/r_D = (n\ell_s)/D$ be the number of elements in a PAT array block.

As in the two-level hierarchy model analysis 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 (the use of more entries in first and second levels 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):

$$h_2 = \log r_D - \log r_M$$

$$h_3 = \log B$$

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 levels from the root to the leaves. This cost, in $t_{s(opt)}$ units, is given by:

$$C_3 = 2(h_2 \times t_{s(mag)} + h_3 \times t_{s(opt)}) = 2(p\,h_2 + h_3)t_{s(opt)} \tag{4}$$

Substituting h_2 and h_3 in Eq. 4 and using the definitions of r_D, r_M and B, we obtain:

$$C_3 = f(n, \ell_s, M, D, p) = 2[\log n + \log l_s + (p-1)\log D - p\log M] \tag{5}$$

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:

$$G_3 = f(n, \ell_s, L_s, M, D) = \frac{C_1}{C_3} = \frac{2\log n - \log L_s + 1}{\log n + \log \ell_s + (p-1)\log D - p\log M} \tag{6}$$

The following example illustrates the improvement obtained with the three-level hierarchy model: Let $n = 50 \times 10^6$ index points, $M = 2$ megabytes, $D = 16$ megabytes, $\ell_s = 30$ characters and $p = 0.03$. Using Eq. (5) and Eq. (1) we obtain: $C_2 = 18\,t_{s(opt)}$ and $C_3 = 13\,t_{s(opt)}$. Thus, the retrieval cost in the worst case is reduced 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 3 shows the results for various text file sizes and compares the performance gain achieved using 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 magnetic medium is a much slower device compared to 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, and (ii) The magnetic disk is 15 to 30 times faster than the optical device.

$M = 1$ megabyte, $D = 16$ megabytes, $\ell_s = 20$ bytes, $p = 0.03$

Text size megabytes	Index points n	Gain G_3 C_1/C_3
10	1.6×10^6	27.7
20	3.2×10^6	15.7
40	6.4×10^6	11.3
80	12.8×10^6	9.0
160	25.6×10^6	7.7
320	51.2×10^6	6.7

Table 3: Performance gain from the three-level hierarchy model

6. EXPERIMENTAL RESULTS

We developed a simulation program to perform the actions of the two-level and three-level models. The simulator maps the text file on the disk sectors and tracks, either magnetic or optical, and computes the exact time needed to access and read any disk position. The input for the simulator is a text file and its SPAT array and PAT array. An amount of main memory is specified for the SPAT array and an adequate value for the SPAT array entry (ℓ_s) is chosen. With these parameters the algorithm scans the PAT array and the text to build the SPAT array.

The performance evaluation of SPAT arrays was obtained by means of a sufficient number of repetitions (for different text sizes) of the following experiment: a set of random search keys is presented to the standard PAT array, the two-level and the three-level searching algorithms. A key can be a word, word prefix or a phrase. For each search key the simulator gives the search time, the number of disk accesses and the performance gain. We present experimental results only for the case in which the answer is bounded by a single PAT array block b. The reason is because the behavior of the algorithm when the answer is bounded by two PAT array blocks follows closely the single case.

The parameters of interest in the simulation are: the text size (in our experiments we used texts ranging from 10 to 160 megabytes), the number of index points in the PAT array (n), the main memory space used by the SPAT (M), the magnetic disk space used by the SPAT (D), the length of each SPAT entry (ℓ_s), the number of elements in each PAT block (b), and the disk cost model, which gives the access time function. We consider an average word length of $\overline{W} = 6$ characters. Thus, the number of index points for a text with $TextSize$ bytes is given by $n = TextSize/\overline{W}$. We assume that the text file is contiguously stored in the disk starting at track 1, and the corresponding PAT array file starts at the first track available after the text file.

6.1. *Two-Level Model*

Table 4 shows the experimental performance gains for the two-level model, for texts ranging from 10 to 160 megabytes, and for SPAT size ranging from 2 to 8 megabytes. We run a set of 400 successful random searches for each text and SPAT size, for CD-ROM disks. For comparison purposes, the same set of queries were searched using the standard PAT array algorithm. The values in Table 4 represent the performance gain, defined by:

$$Gain(time) = \frac{AccessTimePAT}{AccessTimeTwoLevel}$$

and

$$Gain(access) = \frac{NrOfAccessPAT}{NrOfAccessTwoLevel}$$

$M = 2$ megabytes, $r = 100 \times 10^3$, $\ell_s = 20$ bytes, $L_s = 2048$ bytes

Text size (megabytes)	Index points (n)	$Gain(time)$ $\left(\frac{C_1}{C_2} = \frac{AccessTimePAT}{AcessTimeTwoLevel}\right)$	$Gain(access)$ $\left(\frac{C_1}{C_2} = \frac{NrOfAccessPAT}{NrOfAccessTwoLevel}\right)$
10	1.6×10^6	8.5±1.7	8.4±1.7
20	3.2×10^6	7.3±1.4	7.4±1.3
40	6.4×10^6	6.2±1.2	6.4±1.3
80	12.8×10^6	5.6±0.7	5.8±0.8
160	25.6×10^6	5.1±0.6	5.0±0.5

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

Text size (megabytes)	Index points (n)	Gain $\left(\frac{C_1}{C_2} = \frac{AccessTimePAT}{AcessTimeTwoLevel}\right)$	Gain $\left(\frac{C_1}{C_2} = \frac{NrOfAccessPAT}{NrOfAccessTwoLevel}\right)$
10	1.6×10^6	13.3±1.9	12.6±1.4
20	3.2×10^6	10.4±1.3	10.3±1.4
40	6.4×10^6	8.4±1.1	8.5±1.1
80	12.8×10^6	7.8±1.4	7.9±1.4
160	25.6×10^6	6.2±0.8	6.3±0.7

$M = 8$ megabytes, $r = 400 \times 10^3$, $\ell_s = 20$ bytes, $L_s = 2048$ bytes

Text size (megabytes)	Index points (n)	Gain $\left(\frac{C_1}{C_2} = \frac{AccessTimePAT}{AcessTimeTwoLevel}\right)$	Gain $\left(\frac{C_1}{C_2} = \frac{NrOfAccessPAT}{NrOfAccessTwoLevel}\right)$
10	1.6×10^6	17.4±1.9	16.5±1.5
20	3.2×10^6	12.6±1.5	12.4±1.4
40	6.4×10^6	10.9±1.3	10.8±1.1
80	12.8×10^6	8.4±1.5	8.5±1.5
160	25.6×10^6	7.2±0.8	7.4±0.7

Table 4: Experimental results for the performance gain with the two-level model

All values in the table are within a 95% confidence interval. We observe that small files present larger standard deviation. This is because small files takes only a few tracks on the disk, increasing the probability of performing proximal accesses. The difference in cost between proximal and non-proximal access is too large, causing larger variations in the retrieval time.

Figure 10 shows the performance gain plots for the two-level model for $M = 2$ megabytes. The analytical result, indicated by the dashed curve, was obtained from Eq. (3).

6.2. *Three-Level Model*

Table 5 shows the experimental performance gains for the three-level model, for texts ranging from 10 to 160 megabytes, having a SPAT size of 16 megabytes stored in the magnetic disk and a primary index of 1 megabyte stored in the main memory. The values in Table 5 represent the performance gain, defined by:

$$Gain(time) = \frac{AccessTimePAT}{AccessTimeThreeLevel}$$

and

$$Gain(access) = \frac{NrOfAccessPAT}{NrOfAccessThreeLevel}$$

Figure 11 shows the performance gain plots for the three-level model. The analytical result, indicated by the dashed curve, was obtained from Eq. (6).

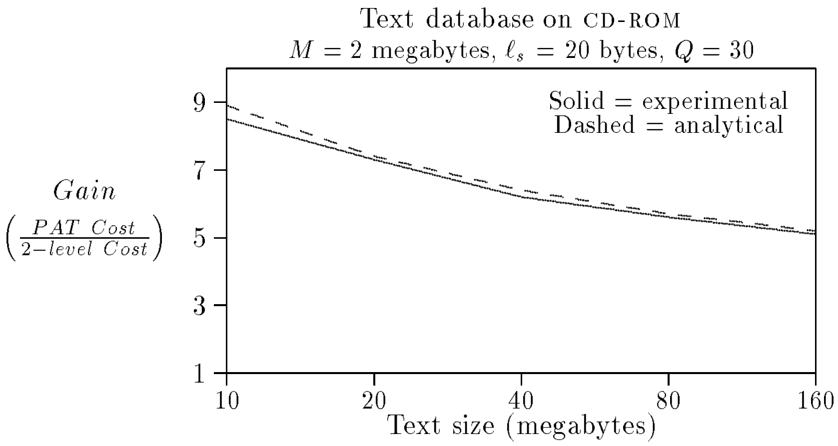

Fig. 10: Performance gain for the two-level model with a SPAT of 2 megabytes

$M = 1$ megabyte, $D = 16$ megabytes, $\ell_s = 20$ bytes, $L_s = 2048$ bytes, $p = 0.03$

Text size (MB)	Index points (n)	$Gain(time)$ $\left(\frac{C_1}{C_3} = \frac{AccessTimePAT}{AcessTimeThreeLevel}\right)$	$Gain(access)$ $\left(\frac{C_1}{C_3} = \frac{NrOfAccessPAT}{NrOfAccessThreLevel}\right)$
10	1.6×10^6	28.5±3.7	28.6±1.2
20	3.2×10^6	14.1±1.6	13.6±1.2
40	6.4×10^6	10.3±1.0	10.5±0.9
80	12.8×10^6	9.0±0.9	8.9±0.9
160	25.6×10^6	7.5±0.8	7.5±0.7

Table 5: Experimental results for the performance gain with the three-level model

6.3. Time to Transfer the SPAT Array from Disk

An important measurement is the time needed to transfer the higher index structures from disk to main memory, since this is an overhead that the user experiments before the hierarchy model can process the first query. Table 6 shows the transfer time for different SPAT sizes, starting from CD-ROM disks and magnetic disks. We reinforce the fact that the three-level model may have the second level, which is the larger index structure in the upper memory levels, permanently stored in the magnetic disk, thus saving transfer time each time a CD-ROM database is used.

Transfer time from disk to main memory $t_x = Seek + Transfer \times SPAT_{size}$ (In seconds)			
SPAT size (megabytes)	CD-ROM @ 600 KB/s	CD-ROM @ 800 KB/s	Magnetic disk @ 5 MB/s
0.5	1.0	0.8	0.1
1.0	1.8	1.4	0.2
2.0	3.6	2.7	0.4
4.0	6.9	5.2	0.8
8.0	13.5	10.2	1.6

Table 6: Time to transfer the SPAT to main memory, before the first query

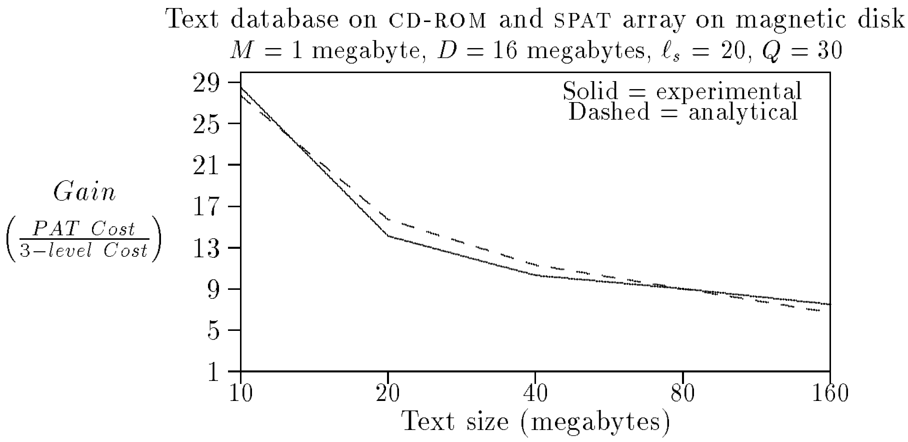

Fig. 11: Performance gain for the three-level model (1 megabyte in main memory)

7. CONCLUSIONS AND FUTURE WORK

We presented two index hierarchies and the corresponding searching algorithms to improve retrieval from full-text databases stored on secondary devices such as magnetic or optical disks. For large text files stored on 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 analyses 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. Performance improvement is achieved with the two-level hierarchy model due to the fact that: (i) the 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 over 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 are determined 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).

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 on 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 file index to access a large number of CD-ROM disks.

We obtained wall-clock times for searching a 279,534,695 bytes text containing three years (1987, 1988, 1989) of the Wall Street Journal†. We used the two-level model to search for the words `sumaria, clock, end, the`. The text was stored in a magnetic disk of a computer with architeture SUN 4M with two processors, 256 megabytes of main memory and running Solaris 2.5. Table 7 presents the search times in milliseconds. Note that the average performance gain of the SPAT model compared to the standard PAT model for the four words is 5.79, which is close to the analytical results obtained for the two-level model (see Table 2).

There are two improvements that can be done to reduce space and time complexities of the models that we have just described. The first one is a compression of the SPAT array index, based upon the lexicographical ordering of the index and the existence of common prefixes in consecutive

†This text is part of the 2 Gigabyte TIPSTER collection [12].

Word searched	Number of occurrences	Search time (milliseconds)			
		Grep	Glimpse	PAT	SPAT
`sumaria`	1	32,400	18,400	60.9	9.25
`clock`	571	30,300	16,500	74.37	13.53
`end`	165,317	24,100	16,600	79.5	16.51
`the`	1,308,083	33,900	17,300	110.05	16.65

Table 7: Wall-clock times for searching a 280 megabytes text file

index entries [4]. The second improvement is related to the reduction of the expected cost of the binary search in the last level of the hierarchy. This reduction of the time complexity is obtained by considering the anticipated knowledge of the expected size of the subproblem produced by reading each disk track. This information is used to devise a modified binary searching algorithm to decrease overall retrieval costs [5].

The PAT array model uses binary search to obtain all occurrences of a string in a text in $O(\log n)$ time. Another alternative for searching an ordered set of keys is interpolation search [10], with $O(\log\log n)$ search time. Interpolation search works similarly like binary search, but in a more sophisticated way: at each step the algorithm makes an interpolation of where the desired key is apt to be in the ordered array. This interpolation is based on the value of the search key and the values of the left and right keys in the current interval in the array. In a main memory setting, interpolation search suffers from a high proportionality constant, due to the need for expensive division and other floating-point operations. In a setting with secondary and tertiary storage, the above costs will be overshadowed by the disk access costs, thus making interpolation search a very promising method to be used with PAT arrays.

Acknowledgements — The authors would like to thank the anonymous referees for their valuable suggestions. We wish to acknowledge Maria Dalva Resende, who helped particularly with the implementation of the algorithm. The authors wish to acknowledge the financial support from the Chilean CONICYT Grant 1950622, the Brazilian CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, IBM do Brasil, Programa de Cooperación Científica Chile-Brasil de Fundación Andes, and Project RITOS/CYTED.

REFERENCES

[1] R. Baeza-Yates, E.F. Barbosa and N. Ziviani. Hierarchies of indices for text searching. In *Proceedings RIAO'94 Intelligent Multimedia Information Retrieval Systems and Management*, pp. 11–13. Rockefeller University, New York (1994).

[2] E. F. Barbosa. *Efficient text searching methods for secondary memory.* Ph. D. thesis, Universidade Federal de Minas Gerais, Brazil, Department of Computer Science, Technical Report 017-95 (1995).

[3] E. F. Barbosa and N. Ziviani. Data structures and access methods for read-only optical disks. In R. Baeza-Yates and U. Manber, editors, *Computer Science: Research and Applications*, pp. 189–207. Plenum Publishing Corp. (1992).

[4] E. F. Barbosa and N. Ziviani. From partial to full inverted lists for text searching. In *Proceedings Second South American Workshop on String Processing*, pp. 1–10, Valparaiso, Chile (1995).

[5] E. F. Barbosa, G. Navarro, R. Baeza-Yates, C. Perleberg and N. Ziviani. Optimized binary search and text retrieval. In P. Spirakis, editor, *Proceedings ESA'95 Third Annual European Symposium on Algorithms*, Springer-Verlag Lecture Notes in Computer Science, v. 979, pp. 311-326, Corfu, Greece (1995).

[6] C. Faloutsos. Signature files. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval Data Structures and Algorithms*, pp. 44–65, Prentice-Hall, Englewoods Cliff, N.J. (1992).

[7] D. A. Ford and S. Christodoulakis. File organizations for optical disks. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval Data Structures and Algorithms*, pp. 83–101, Prentice-Hall, Englewoods Cliff, N.J. (1992).

[8] G. H. Gonnet. Unstructured data base or very efficient text searching. In *Proceedings of the Second ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pp. 117–124, Atlanta, Georgia (1983).

[9] G. H. Gonnet. *Pat 3.1: An efficient text searching system.* Center for the New Oxford English Dictionary. University of Waterloo, Canada (1987).

[10] G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures.* Addison-Wesley, Reading, Mass., second edition (1991).

[11] G. H. Gonnet, R. Baeza-Yates and T. Snider. New indices for text: Pat trees and Pat arrays. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval Data Structures and Algorithms*, pp. 66–82, Prentice-Hall, Englewoods Cliff, N.J. (1992).

[12] D. Harman. Overview of the third text retrieval conference. In *Proceedings Third Text Retrieval Conference (TREC-3)*, pp. 1–19, National Institute of Standards and Technology Special Publication 500-207, Gaithersburg, Maryland (1995).

[13] D. Harman, E. Fox, R. Baeza-Yates and W. Lee. Inverted files. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval Data Structures and Algorithms*, pp. 28–43, Prentice-Hall, Englewoods Cliff, N.J. (1992).

[14] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, Mass. (1973).

[15] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *ACM-SIAM Symposium on Discrete Algorithms*, pp. 319–327 (1990).

[16] U. Manber and S. Wu. Glimpse: a tool to search through entire file systems. Technical Report 93-34, Department of Computer Science, The University of Arizona, Tucson, Arizona (1993).

[17] D. R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, **15**(4):514–534 (1968).

[18] G. Navarro. An optimal index for Pat arrays. In N. Ziviani, R. Baeza-Yates and G. Guimarães, editors, *Proceedings Third South American Workshop on String Processing*, Carleton University Press International Informatics Series, v. 4, pages 214–227, Recife, Brazil (1996).